Louisiana State University
LSU Digital Commons

LSU Master's Theses | Graduate School

10-22-2018

Non-Parametric Classification of Time Series Using Permutation Ordinal Statistics

Aldo Duarte Vera Tudela
Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_theses

 Part of the Signal Processing Commons, and the Systems and Communications Commons

Recommended Citation
Duarte Vera Tudela, Aldo, "Non-Parametric Classification of Time Series Using Permutation Ordinal Statistics" (2018). *LSU Master's Theses*. 4815.
https://digitalcommons.lsu.edu/gradschool_theses/4815

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.

NON-PARAMETRIC CLASSIFICATION OF TIME SERIES USING PERMUTATION ORDINAL STATISTICS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Aldo Duarte Vera Tudela
B.S. in Telecomunication Enginering, Pontificia Universidad Catolica del Peru, 2012
December 2018

Acknowledgments

First of all, I would like to sincerely thank Dr. Shuangqing Wei for his insightful guidance and seemingly unbounded enthusiasm throughout the course of my graduate studies. Not only guiding me through research and pointing the areas I had to strength, but as a friend emotionally supporting me during some rough personal circumstances. I would also like to thank my other thesis committee members, Dr. Xiangwei Zhou and Dr. Mingxuan Sun, for devoting their time to help me with my thesis and consider many issues that I had previously neglected before the onset of this thesis work.

I would like to thank my parents, extended family, and friends for giving all their support and always believing in me even in the most difficult moments of my studies. I would have not make it through without their support.

Table of Contents

List of Tables

List of Figures

Abstract

The present thesis explores some approaches to classify time series without prior statistical information using the concept of permutation entropy. Motivated by the results from a previous published and relevant work that set similarity relationships between EEG time series, a reproduction of the proposed approach was performed giving negative results. The failure to reproduce those results led to the conclusion that the approach of building statistics from permutation patterns have to be complemented with another metric in order to be used for classification purposes. The concept of Total Variation Distance (TVD) was then used to develop three algorithms to classify time series in a non-parametric way.

At first, the developed algorithms were tested using EEG time series. Even though the results using the developed algorithms were better than previous results, they were not as satisfactory as desired. However, the inherent complexity of brain measurements led to switch to self-generated data to test the algorithms. Using time series coming from different sets of filtered versions of Gaussian white noise the classification was performed. For comparison purposes a parametric classification approach using the Maximum Likelihood Estimation was also used. Results showed that when each set of data came from the same filtering equation the classification using the developed algorithms was optimal reaching 100% success rate in many cases, being as good as the ML approach. On the other hand, when each set of data came from a mixture of different filter equations that generate the time series (reflecting the complex situations we faced when processing EEG data) , results were fairly successful with variations with respect to the ML approach, which was outperformed in some cases but also not surpassed in others.

The results obtained pointed the permutation entropy analysis to be an approach in the right direction to efficiently classify time series, however more research needs to be done to adjust the correct metric to get better results.

Chapter 1

Introduction

Information is an essential aspect in today's world. Every minute millions of different types of data are generated, stored and shared. From important radio navigation signals to the last gossip between your friends, information being crucial or just entertaining is a vital aspect of humanity these days. As presented in an article in Forbes by May 2018, only in internet activities are 2.5 quintillion bytes of data created each day [2]. And this daily value is not considering the huge of amount of data created in navigation, satellites, cell phones systems, and many other sources of information. Humanity wants to access and create data all the time. According to the same article every second Google processes more than 40,000 searches, and every minute of the day Snapchat users share 527,760 photos, more than 120 professionals join LinkedIn, users watch 4,146,600 YouTube videos, 456,000 tweets are sent on Twitter, and Instagram users post 46,740 photos. These numbers with the new trend of Internet of Things (IoT) and new apps being developed every year will be even higher in the upcoming years.

More data generation will require better devices to access and create that data, and most of such devices use the air as their channel to propagate their messages. The spectrum is becoming the most important information channel and in the following years its usage will keep growing. It is estimated that by 2020 the number of smartphones users will increase up to 2.87 billion people [3], and this is not considering other types of devices like tablets or laptops. Therefore, the rapidly growing trend will demand even more dense spectrum traffic. For that reason, more intelligent and efficient ways to use this channel have to be developed. One possible implementation for that purpose is to have smart radio elements that are able to scan the spectrum and make decisions according to the situation in terms of interference, traffic and channel contention. However, the biggest issue with this idea is that each time the air interface is scanned what we get is raw data in form of time series and there is no way to identify if we are dealing with information or interference at first sight. So, to achieve this some method have to be developed to give devices the way to learn from its environment using just the raw data scanned from the spectrum. This raw data comes in the form of time series. Therefore, in order to be able to overcome this challenge the first step is to address a broader issue, that is the mining of underlying statistics in time series.

Time series have an structure that produce correlation between its components. This property intuitively let us believe that there is a way to characterize the underlying statistics producing the time series by analyzing its structural behavior. The challenge is to extract relevant statistics from the time series that could be useful for other purposes. This thesis aims to gives a first step to overcome the already stated challenge by exploiting a

recently proposed non-parametric method to classify time series for certain scenarios. It was considered at the beginning of the study the resulting approach should meet the following requirements:

- Being able to build statistics only using a set of time series corresponding to some measurement or experiment.
- With the statistics built define scenarios that will be used for classification.
- Being able to classify new time series according to the defined scenarios, and successfully identify which case they belong to.
- Try to make the algorithm as easy to compute as possible.

The rest of this chapter will give more insight of the motivation of the method adopted as well as the presentation of the concepts used in developing our algorithms.

1.1 Time Series

The very first step to develop any approach is to identify the type of data that will be processed. Since the ultimate goal of the study is to be able to use it in communication scenarios using the spectrum as channel, the data scanned will be taken over a period of time and the order of this data is important since the observation is constraint by time. This essentially entails a time series structure, so understanding the properties of this type of data is key for all the future analysis.

In simple words, a time series is a sequence of data points being recorded at specific times. Formally, let (Ω, Γ, P) be a probability space, and T an index set. A real valued stochastic process is a real valued function $X_t(\omega)$ such that for each fixed $t \in T$, $X_t(\omega)$ is a random variable on (Ω, Γ, P). When the index set T corresponds to time indices (discrete or continuous), we will call $X_t(\omega)$ a time series. For fixed ω, $X_t(\omega)$ is a real valued function of t that is called a realization of the time series. When we look at the plot of a recorded time series, we are looking at one realization out of the collection of all possible realizations. We typically suppress the ω and write X_t. For a discrete time series, the set of times T is a discrete set, and the measurements are typically at successive times spaced at uniform intervals. Continuous time series are obtained when observations are recorded over some time interval, e.g. (0,1) [4].

Examples of time series are:

- Minimum daily temperatures over 10 years.
- Electroencephalogram (EEG) data.

- Daily closing stock prices.
- Weekly interest rates.
- Sales figures.
- Daily female births in California in 2012.

Among innumerable other sequences based on industrial, economic, and social phenomena, and studies in medicine, geophysics, and engineering.

An important property of time series that is useful for our purpose is the fact that the points forming the time series are not completely independent between each other. There is a correlation among the measurements and it is often said that the past can influence the future in a given time series variable. This property is called memory, and relates to the correlation existing between different points in a timeline. This concept is useful in statistics because the strength of such correlation has a huge impact on the predictability of the data. One distinction between time series is related to the rate of decay of statistical dependence of two points with increasing time interval or spatial distance between the points, resulting in time series with short and long memory. One way of characterizing long and short memory time series is in terms of their autocovariance functions. For a short case, the coupling between values at different times decreases rapidly as the time difference increases. Either the autocovariance drops to zero after a certain time-lag, or it eventually has an exponential decay. In the case of the long memory case, there is much stronger coupling, the decay of the autocovariance function follows power-law, and is much slower than an exponential rate. [5].

Going into detail about how to quantify the amount of memory within a time series is not part of our study. What is relevant to our study is that that time series have a memory that can be used to build or model the data underlying statistics. Therefore, the next step was to discover one way to take advantage of that feature. Thus, some statistical model or technique will be chosen to extract the inherent data statistics dwelling within the time series.

1.2 Statistical Models to Classify Time Series

In order to develop a classifying algorithms some statistical models and assumptions shall be adopted. The more intuitive types of modeling are the ones that follow some a priori assumptions and rely on known probabilistic functions. Therefore, at first we will examine approaches using parametric models.

1.2.1 Parametric Models

A parametric model is a model where it is assumed that the data is an observation of a random vector $\overrightarrow{x} = (x_1, x_2, ..., x_n)$, and the joint distribution of $\overrightarrow{x}$ is a member of some family of distributions that depend on a small number of unknown parameters. Formally, is a collection of probability distributions such that each member of this collection, P_θ is described by a finite-dimensional parameter θ. The set of all allowable values for the parameter is denoted $\Theta \in R^k$, and the model itself is written as [6]:

$$P = \{P_\theta | \theta \in \Theta\}$$

When the model consists of absolutely continuous distributions, it is often specified in terms of corresponding probability density functions [6]:

$$P = \{f_\theta | \theta \in \Theta\}$$

Therefore, in this type of model it is assumed that the time series obeys a specific probability distribution. For instance, if it is assumed that the time series follows a family of Gaussian distributions ($f_Y(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} exp(-0.5(x-\mu)^T)\Sigma^{-1}(x-\mu))$), where Σ is the covariance matrix and μ the mean vector, then some scenarios can be built using covariance matrices performing the well known Maximum-Likelihood approach. This parametric model will be revisited and further explained in chapter 3.

Parametric models had been used before for classification purposes. Even though, there are well known and defined parametric models used in classification approaches, there is still recent research going on for classification of time series using parametric models. In [7] a new approach for Multivariate Time Series classification, using a parametric derivative dynamic time warping distance (DTW), is proposed. In that work modifications to DTW as a more accurate time series similarity measure was presented, as well as a template-based approach for Human Activity Recognition. Similar approach was used in [8] where the DTW concept is used again in the nearest neighbor (1NN) classification method in the case of both univariate and multivariate time series analysis. Among more classical approaches there is the one in [9] where trends in seasonal data were extracted using penalized likelihood and parametric bootstrap.

Despite that parametric models are usually more precise and reliable than other types of models, for the algorithm we want to develop it is not possible to assume any specific distribution ruling our data.The generality desired for our proposed approach requires no presumed specific distributions on the data, nor even mixture distributions. Thus, the modeling has to be done in a non-parametric way.

1.2.2 Non-Parametric Models

A non-parametric model is a model where it is assumed that the data is an observation of a random vector $\overrightarrow{x} = (x_1, x_2, ..., x_n)$, and the joint distribution of $\overrightarrow{x}$ lives in a very large family of distributions not described parametrically. In this type of models there is no constraint in the distribution generating the data. They are usually less precise and powerful than its parametric counterparts but because they make fewer assumptions, their applicability is much wider. Non-parametric models differ from parametric models in that the model structure is not specified a priori but is instead determined from data. The term non-parametric is not meant to imply that such models completely lack parameters but that the number and nature of the parameters are flexible and not fixed in advance [6].

Examples of non-parametrical models are:

- AndersonDarling test: tests where a sample is drawn from a given distribution.
- Statistical bootstrap methods: estimates the accuracy/sampling distribution of a statistic.
- Wilcoxon signed-rank test: tests whether matched pair samples are drawn from populations with different mean ranks.
- Kendall's tau: measures statistical dependence between two variables.
- Median test: tests whether two samples are drawn from distributions with equal medians.
- TukeyDuckworth test: tests equality of two distributions by using ranks.

There are several research going on for classification purposes using non-parametric models. Specifically for time series, one recent work is the one presented in [10] where a latent source model for time series was proposed, which naturally leads to a weighted majority voting classification rule that can be approximated by a nearest-neighbor classifier. This classifier was developed to try to anticipate trending words in platforms like Twitter. Another research in [11] presents a method for non-parametric trend modeling to use such trends in machinery health prognostics to develop a method that can extract features representing the nominal behavior of the monitored component. Also, in [12] a non-parametric approach to identify similarities in a set of simultaneously observed time series was developed using polynomial regression and classified according to standard clustering procedures. Among the newest approaches there is [13] where a weighting mechanism that, coupled with convolutional networks forms a new neural network architecture for time series prediction, designed for regression tasks on asynchronous signals in the presence of high amount of noise.

The presented related works are only a little share of all the research done using non-parametric models with time series. It was encouraging to see that there are several approaches taking advantage of the time series memory with a fair success in their performance.

However, some non-parametric procedures are really complex in its concept and computation while our objective is to develop a not too complex and light computational algorithm. For that reason, one rising and novel approach caught our attention because of simplicity and potential to fit into the desired goal. This concept is the Permutation Entropy.

1.3 Permutation Entropy

The concept of Permutation Entropy was first presented in 2002 by Christoph Bandt and Bernd Pompe. The novelty of the approach is that it builds statistics according to the orderings or permutations presented in the data in specific windows. Formally, Permutation Enthropy as defined in [14] is an entropy measure that quantify information based on the occurrence or absence of certain permutation patterns in a time series. It is defined by:

$$H_n = -\sum_{j=1}^{n!} P'_j log_2(P'_j)$$

where P'$_j$ represent the relative frequencies of the possible permutations, and n is the window length that will determine the number of items in each permutation where the maximum possible number of permutations is $n!$. For instance, if n = 3 then the first three items (x_1, x_2, x_3) will be selected and its permutation pattern analyzed, the window then shift one slot so the the next permutation consider the items (x_2, x_3, x_4).

At each time s of a given a time series $\chi = \{x_t : t = 1, .., N\}$ a vector composed of the n-th subsequent values is constructed:

$$s \mapsto (x_s, x_{s+1}, .., x_{s+(n-2)}, x_{s+(n-1)})$$

where n is called the embedding dimension, and determines how much information is contained in each vector.

To this vector, an ordinal pattern is associated, defined as the permutation $\pi = (r_0 r_1 ... r_{n-1})$ [15]. The idea behind permutation entropy is that patterns may not have the same probability of occurrence, and thus, that this probability may unveil relevant knowledge about the underlying system.

Permutation Entropy can be further extended by considering an embedding delay τ [15]:

$$s \mapsto (x_s, x_{s+\tau}, .., x_{s+\tau(n-2)}, x_{s+\tau(n-1)})$$

when τ is greater than one, the values composing the permutations are taken non-consecutively, thus mapping the dynamics of the system at different temporal resolutions.

For a better understanding we next present an algorithm to compute the permutation entropy as it was defined in [16]:

- Define the order of permutation n. That leads to the possible permutation patterns $\pi_j(j = 1, .., n!)$ which are built from the numbers 1, ..., n..
- Initialize i = 1 as the index of the considered time series $\{x'_i\}, i = 1, ..., N$ and the counter $z'_j = 0$ for each permutation.
- Calculate the ranks of the values in the sequence $x_i, ..., x_{i+n-1}$ which leads to the rank sequence $r_i, ..., r_{i+n-1}$. The ranks are the indices of the values in ascending sorted order.
- Compare the rank sequence of step 3 with all permutations pattern and increase the counter of the equal pattern $\pi_k = r_i, ..., r_{i+n-1}$ by one $(z_k = z_{k+1})$.
- If $i \leq N$ then increase i by one $(i = i + 1)$ and start from step 3 again. If $i > N - n$ go to the next step. (N is the total number of points in the time serie)
- Calculate the relative frequency of all permutations π_j by means of $p'_j = z_j/\Sigma z_k$ as an estimation of their probability p_j.
- Select all values of p'_j greater than 0, and calculate the permutation entropy using its definition equation.

It is highly recommended to set a value of n such that the relation $N > 5n!$ is met, which constrains n to not be greater than 7 for most applications.

The original approach proposed in [14] do not allow the presence of equal values in a time series. In order to handle this situation there are three strategies [16]:

- The ranks of these values are determined in accordance to their order in the sequence.
- The identity is eliminated by adding white noise with the strength of the stochastic term being smaller than the smallest distance between values.
- The values get the same rank number within the regarded sequence, for example 3, 6, 2, 3, 8 leads to the rank sequence 2, 4, 1, 2,5.

An implementation for MATLAB of the PE algorithm, including the different mentioned strategies to overcome the problem of equal values, can be found in [17].

This information measurement is still subject of research. Among the most related conducted research to our approach we have the work done in [18] where a test for independence between time series was performed by using symbolic dynamics and permutation entropy as a measure of serial dependence. Also in [19] an approach based on the estimation of the permutation entropy combined with an intensive complexity measure, building up the complexity entropy causality plane, was used to distinguish and discriminate songs. A new concept in [20] is introduced PE/WPE technique to multiple time scales, called multiscale permutation entropy (MSPE)/multiscale weighted-permutation entropy (MSWPE), which are applied to investigate complexities of different traffic series. Interesting results were achieved in that research where the approach successfully detected the temporal structures of traffic signals and distinguish the differences between workday and weekend time series. Another interesting discovery using the permutation entropy is shown in [21] where it was observed that in electroencephalogram (EEG) analysis, value changes of PE correlate with clinical observations, among them the onset of epileptic seizures or the loss of consciousness induced by anesthetic agents. Finally, a recent study in [22] showed that equal values in the time series depending of its frequency of appearance could lead to false conclusion when performing the permutation entropy analysis. This observation was taken into consideration during the research, luckily the time series used had a really low probability to have equal values.

As seen, permutation entropy is motivating several research involving time series. Nevertheless, there is one specific work that caught our attention involving the permutation entropy concept used for similarity relationships between time series. This proposed idea motivated the attempt of a reproduction, and while attempting it the idea for our own classification algorithm came to light. Further explanations about it are presented in the next chapter.

1.4 Organization of the Thesis

In the following chapters will be presenting our work and the results obtained. The steps followed in the research will be presented, as well as the motivation for the decisions made when moving to a next step. Our study will prove that the permutation entropy is indeed a good approach to characterize underlying statistical behavior in time series. Moreover, we developed three algorithms to classify time series based in the concepts of permutation entropy and total variation distance. The proposed algorithms had a fair success rate in its classification purposes, giving a foundation for more research in the use of the PE concept for non-parametric time series classification.

The thesis is structured as follows, Chapter 2 presents a previous developed approach used as starting point whose reproduction results had taught us valuable lessons that helped to model the final proposed algorithms. Chapter 3 presents the developed algorithms and showed several different testing scenarios. Finally, chapter 4 shows the future works that

can be done with this starting point, and the lessons learned while making the study as a manner of final conclusion.

Chapter 2

Application Using PE

As explained in the previous chapter, the goal is to develop a classification algorithm in a non-parametric way using the concept of Permutation Entropy (PE). In this chapter, an interesting previous work is reviewed where the concept of PE is used to determine similarity relationships between time series. First, the previous work done is presented and explain. Then, a reproduction of it was performed and the results presented. The outcome of the reproduction led to develop alternative variations of the original work, those variations are presented as well as the results. Lastly, an alternative metric is proposed to enhance the approach

2.1 Similarity Approach Using PE

In [1] a relevant usage of PE is presented. The paper's goal is to estimate "the bias, variance and confidence intervals for the Permutation Entropy estimation, along with hypothesis testing, that consists in simulate bootstrap symbolic time series samples that are thought to be produced by a probabilistic model with a fixed transition probability extracted from the original time series". Essentially, this work proposed that using one time series of one experiment of a given scenario and calculating its PE, we can then apply an iterative bootstrap approach with an estimator of the PE in order to simulate more experiments and therefore get statistical measurements.

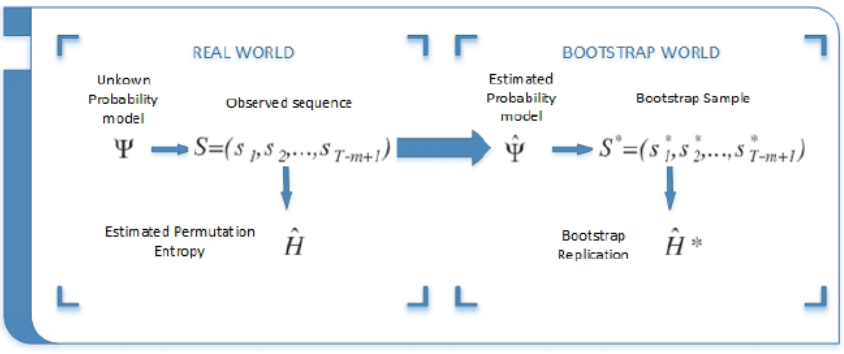

Figure 2.1: Bootstrap Approach [1]

With this estimated distribution the variance, the bias and the confidence intervals of the estimator can also be obtained. The metric used is the Permutation Entropy so an estimator of it must be obtained. The way to reproduce new sequences lies within the concept of Transition Probability Matrix that is calculated during the permutation analysis.

2.1.1 Permutation Entropy Estimator

For a time series $\{X_t\}_{t\in T}$ the permutation entropy is calculated with a dimension m leading to a finite random process $\{S_t\}_{t\in(T-m+1)}$ with $S_t \in S_m = \{\pi_1, \pi_2 ... \pi_{m!}\}$ for all posible $m \geq 2$. This realization of the symbolic sequence is thought to be produced by a probabilistic model with a fixed transition probability P^{ij} denoted $\Psi(P^{ij})$. The model $\Psi = \Psi(P^{ij})$ is estimated to bootstrap the Permutation Entropy. The computation of the PE will lead to the following relative probabilities according to [14]:

$$\hat{P}_T(\pi_i) = \frac{n_i}{T - m + 1}$$

With n_i being the number of times state π_i is observed up to time T-m+1. Then the transition probabilities of the symbol sequence will be defined as:

$$P^{ij} = P(S_{t+1} = \pi_j | S_t = \pi_i)$$

Since it is not known the exact value of the transition probabilities, the following estimator is used:

$$\hat{P}_T^{ij} = \begin{cases} \frac{n_{ij}}{n_i}, & \text{if } n_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

n_{ij} = number of transitions observed from π_i to π_j up to time T-m+1.

By the law of total probability we have that:

$$P(\pi_j) = \sum_{i=1}^{m!} P(\pi_i) P^{ij}$$

Leading to the conclusion that the estimator $\hat{P}_T(\pi)$ is determined by the estimation of P^{ij}. Therefore the estimation of the permutation entropy will be:

$$\hat{H}_T = \left\{ -\sum_{i=1}^{N} \hat{P}_T(\pi_i) \ln \hat{P}_T(\pi_i) \right\}$$

The empirical statistics developed gives a transition probability matrix which is used to simulate new bootstrapped sequences. In [1] the algorithm to perform the bootstrap for the the estimated PE is presented.

2.1.2 Method to compare two time series using Permutation Entropy and Bootstrap

A confidence interval for the difference between the permutation entropy of two different time series can be made. In inferential statistics there exists a direct relationship between confidence intervals and hypothesis testing. A two-sided (1-α) confidence interval in the difference between two measures can be used to determine if those two measures are significantly different by only checking if the zero belongs to this interval, since in [1] the hypothesis used to test similarity between time series is that their PE should be equal. However,even if two time series share the same permutation entropy, we only have checked a necessary condition under which two series are equivalent in statistical sense.

$$H_0 : \Delta = H_1 - H_2 = 0$$

If $0 \notin (1-\alpha)100\% CI(\Delta)$ the reject H_0 and

$$H_1 \neq H_2$$

The Algorithm of the method proposed in [1] is presented as follows:

- Compute $\hat{H}_{1T}$ for the first time series.
- Compute $\hat{H}_{2T}$ for the second time series.
- Compute $\hat{\Delta} = \hat{H}_{1T} - \hat{H}_{2T}$.
- while $i \leq B$ do (B = Number of bootstrapped iterations defined by user)
  generate $H_1^*(i)$ (i^{th} bootstrapped value of PE generated from first time series)
  generate $H_2^*(i)$ (i^{th} bootstrapped value of PE generated from second time series)
  end while
- for i in 1 to B do
  for k in 1 to B do
  compute $\Delta_T^*(n) = \hat{H}_{1T} - \hat{H}_{2T}$

end for
end for

- Compute $\Delta_T^*(\bullet) = \frac{1}{B^2}\sum_{i=1}^{B^2}\hat{\Delta}_T^*(n)$
- sort $\delta^*(n) = \Delta_T^*(n) - \hat{\Delta}_T^*(\bullet)$ in increasing order
- Set confidence level $1-\alpha$.
- Compute $\left\{\delta^*_{\alpha/2} \middle/ \frac{\#(\delta^* < \delta_{\frac{\alpha}{2}})}{B} \leq \frac{\alpha}{2}\right\}$
  (For example, if B=1000 and $\alpha = 10\%$ then $\delta^*_{\alpha/2}$ is the 950^{th} value in $\delta^*(n)$)
- Compute $\left\{\delta^*_{(1-\alpha/2)} \middle/ \frac{\#(\delta^* < \delta_{(1-\frac{\alpha}{2})})}{B} \leq 1-\frac{\alpha}{2}\right\}$
  (For example, if B=1000 and $\alpha = 10\%$ then $\delta^*_{1-\alpha/2}$ is the 50^{th} value in $\delta^*(n)$)
- The lower bound of the confidence interval is $\hat{\Delta} + \delta^*_{\frac{\alpha}{2}}$
- The upper bound of the confidence interval is $\hat{\Delta} + \delta^*_{(1-\frac{\alpha}{2})}$
- If 0 does not belong to the interval then $H_1 \neq H_2$ with α level of signification.

2.1.3 Data Used

Five different sets of EEG data was used. Set A and Set B for healthy patients, and Set C and Set D for epileptic patients. Each one of those sets have 100 time series. Each time series in every set has a length of 4097 data points with a sampling frequency of 173.61Hz. For testing purposes, 10 time series from each set were selected at random.

The time series correspond of brain activity measurements and the different sets were defined as:

- Set A: Surface EEG recordings from five healthy volunteers in an awake state with eyes open.
- Set B: Surface EEG recordings from five healthy volunteers in an awake state with eyes closed.
- Set C: Intracranial EEG recordings from five epilepsy patients during the seizure free interval from outside the seizure generating area.
- Set D: Intracranial EEG recordings from five epilepsy patients during the seizure free interval from within the seizure generating area.
- Set E: Intracranial EEG recordings from five epilepsy patients during a seizure.

Set E was not used in [1] but it was included in our reproduction. Details about the recording technique of these EEG data can be found in [23].

2.1.4 Expected Results

We already defined the algorithm to carry on the reproduction. Now it is time to explain a little more insight of the decision making and show the anticipated results presented in [1]. In order to define the similiarity, after performing the bootstrap iterations of the PE a 90% Confidence Intervals for the 10 EEG signals of brain activity for different groups and recording regions are performed. The overlapping between intervals does not necessarily means that there are no significant differences between the two Permutation Entropies. To reach that conclusion, a hypothesis test for the difference must be made as defined in 2.1.2.

For instance, a test was performed for difference in the Permutation Entropy between the 10 EEG signals of healthy volunteers in an awake state with eyes open (SetA) and the 10 EEG signals of healthy volunteers in an awake state with eyes closed (SetB). Each EEG signal of SetA was compared with each signal of SetB with a 10% significance level, and the conclusion anticipated in [1] is that the differences seems to be at random, indicating that is no real difference between these two types of EEG signals.

The same analysis is extended to all the different types of patients. While the differences between SetA and SetB seems to be at random, all EEG signals of those Sets are different in every test to the EEG signals of SetC and SetD. Instead, between SetC and SetD again the differences are distributed between significant and not significant. The anticipated results are shown in the Figure 2.2:

where a green square means a positive result of the null hypothesis, and a red square a negative one.

With our own generated code the anticipated results will be tested along with some confidence own generated data. More detailed of the the generated code for this part can be found in Appendix A.

2.2 Implementation Test and Results

In this section the approach proposed in [1] is tested. Before reproducing the method using the EEG data some confidence building test were done to prove that the code developed works as expected.

For every experiment running our own generated algorithms we always considered values of n = 3, $\tau = 1$, and confidence interval of 90%. Even though this code is flexible allowing to

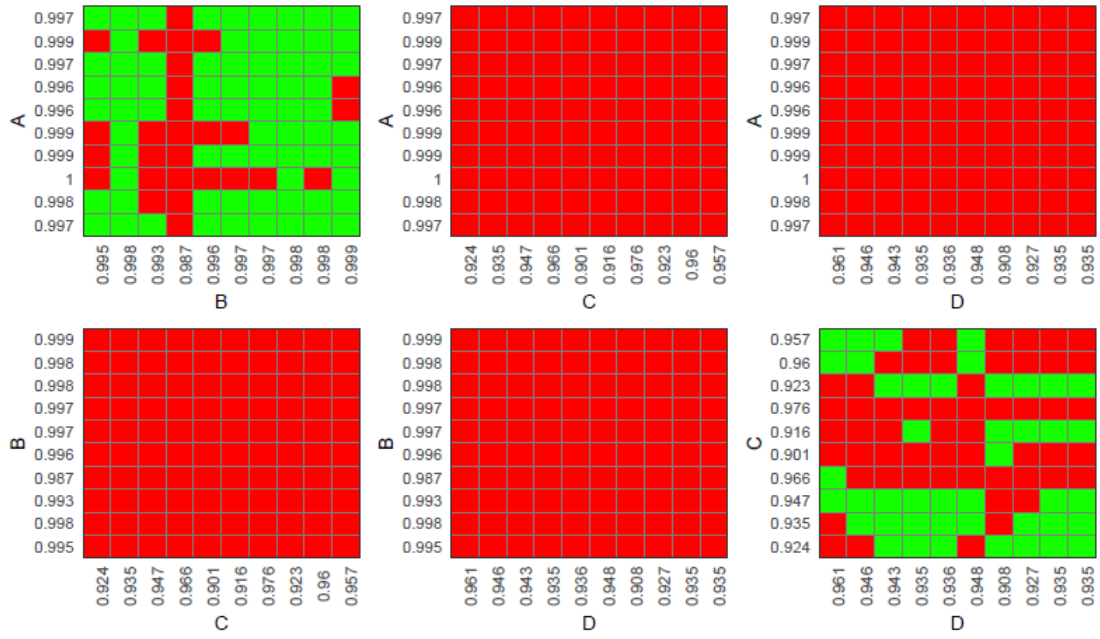

Figure 2.2: Comparison Results [1]

get accurate results for different parameters, we set those ones to have a better reproduction of the work done in [1] and a value $n = 3$ facilitates the analysis process. Some further analysis analysis of the underlying processes taking place in the permutation analysis when $n = 3$ are presented in Appendix B.

2.2.1 First Test: Gaussian sequences

The first test to the proposed PE with bootstrap algorithm was to create our own data to anticipate obvious results. Four different sets each with 10 times series containing 2000 points, named group X, Y, Z and W. Groups X and Y are random white Gaussian noise points while Z and W are filtered versions of groups X and Y (Z from Y and W from X). A FIR filter was used with transfer equation $Y_n = \alpha_0 X_n + \alpha_1 X_{n-1}$. Intuitively, the algorithm should show similarity when comparing sets X with Y, and Z with W, otherwise a dissimilarity should be displayed. Running the comparisons lead to the following results:

The results were as expected, the algorithm successfully identified the similarity between groups X and Y, and Z and W, while showing complete dissimilarity when any other pair of groups are compared.

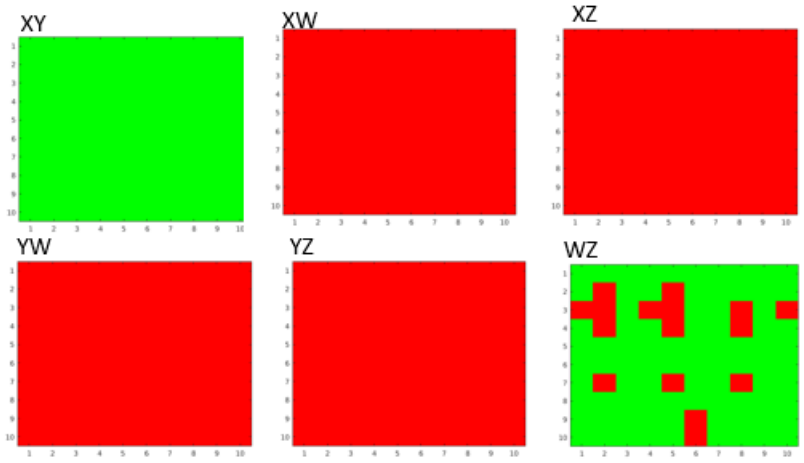

Figure 2.3: Gaussian and Filtered Generated Data Comparison Results

2.2.2 Second Test: Alternative Gaussian sequences

A second test using own generated data was performed. The scenario is similar to the previous one but this time group X are time series coming from a zero mean, unit variance Gaussian distribution, and group Y from a 150 mean, 4 variance Gaussian distribution. Four coefficients were defined, being $\alpha_1 = 0.6$, $\alpha_2 = 0.4$, $\alpha_3 = 0.8$, and $\alpha_4 = 0.2$. Group W is the filtered version of group X using coefficients α_1 and α_2, while group Z is the filtered version of group Y using coefficients α_3 and α_4. Running the algorithm resulted in:

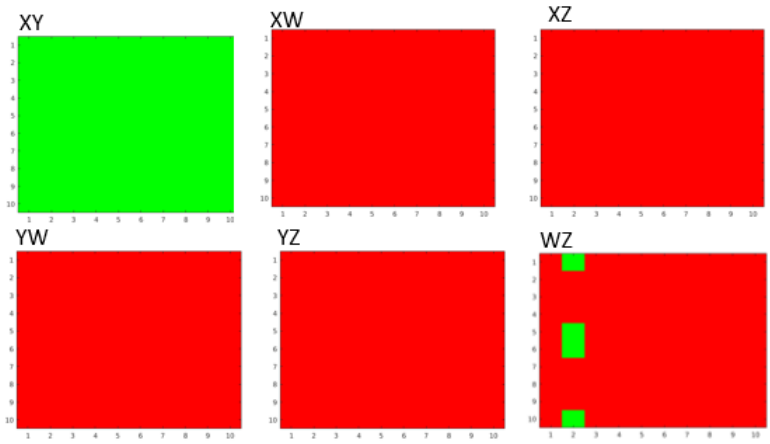

Figure 2.4: Alternative Gaussian and Filtered Generated Data Comparison Results

Once again, the results are still as predicted. Since the coefficients of the filters are different, differences between group Z and W were expected.

2.2.3 Third Test: Markov Chains

For this test four transition probability matrices were defined:

$$TPM1 = \begin{bmatrix} 0.5 & 0.375 & 0.125 \\ 0.25 & 0.125 & 0.625 \\ 0.25 & 0.375 & 0.375 \end{bmatrix} TPM2 = \begin{bmatrix} 0.5 & 0.250 & 0.250 \\ 0.25 & 0.1 & 0.7 \\ 0.2 & 0.4 & 0.4 \end{bmatrix}$$

$$TPM3 = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} TPM4 = \begin{bmatrix} 0.7 & 0.15 & 0.15 \\ 0.1 & 0.65 & 0.25 \\ 0.15 & 0.3 & 0.55 \end{bmatrix}$$

Using the estimator concept time series were generated from the defined TPM's. Where group X had time series created from TPM1, group Y from TPM2, group W from TPM3, and Z from TPM4. As we can see in the construction of the matrices TPM1 and TPM2 are similar matrices and so TPM3 and TPM4. The results of this test were:

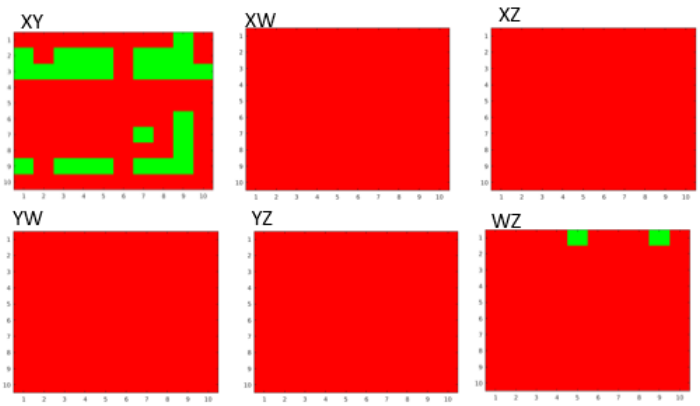

Figure 2.5: Markov Chains Generated Data Comparison Results

The results were not entirely as planned. Between groups X and Y there was some similarities found, for the case of groups W and Z just two cases showed the expected relationship. The dissimilarity relationships were portrait as expected. It is still safe to say that the algorithm is working the way it should be and is ready to test the anticipated results for the EEG data.

2.2.4 Reproduction Using the EEG Data

Now that we built a good confidence that our algorithm works properly it was time to try to reproduce the work done in [1] using the same data and procedure described in Section 2.1.4. The results obtained were:

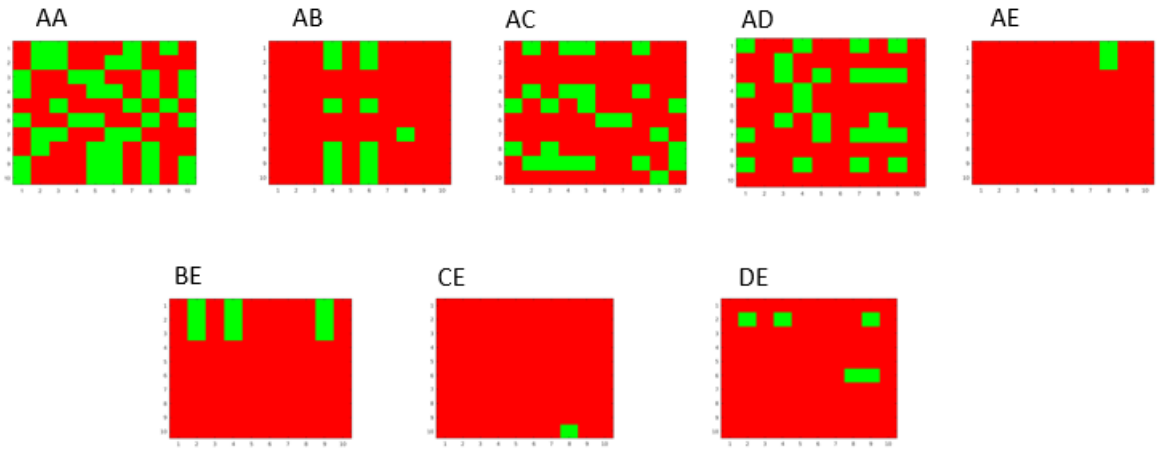

Figure 2.6: Comparison between EEG Data

Where we can see that the obtained results were not as expected. In Figure 6 we can see that setA has more similarity with C and D than with setB, completely differing with the claims presented in the cited paper. The presented results are only a small portion of all possible comparisons between each group however, they showed the overall trend since the results between any other groups of ten would present similar outcomes. Running the complete 100 by 100 comparison between each group was not possible because the algorithm took an excessive amount of time without giving any result. Therefore, we used 10 by 10 comparisons resulting in 100 graphics for each analysis between sets.

2.2.5 PE Histograms

As seen in Section 2.2.4 our algorithm was not able to reproduce the results anticipated in [1]. Up to this point we were focused in trying to get the desired results but did not stop to question the results we were chasing or to analyze the data we were using. Since the decision making for the comparison method depends on the PE, we calculated the PE for each time series in all the data sets giving as result the following histograms:

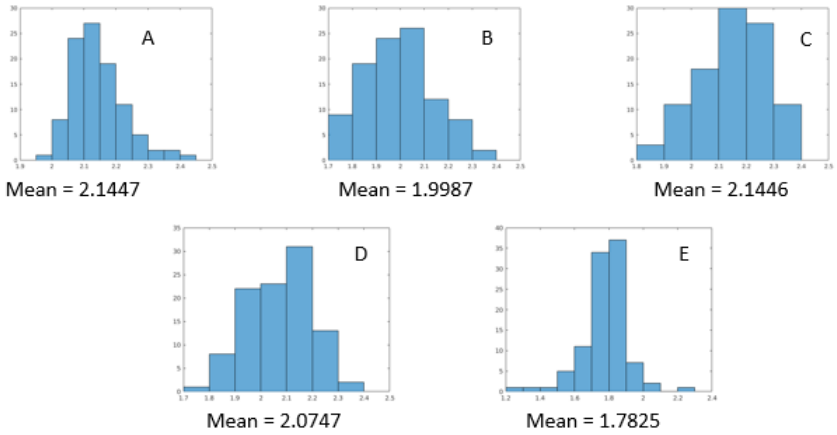

Figure 2.7: PE Histograms

This results gave the following conclusions:

- We did not really understand the nature of the data what we were working with.
- PE is a measure pointing how random is the information we are dealing with, therefore this value will point that in the EEG measurements.
- Defying our intuition, the PE value is similar between groups A, C, D, then decrease for group B, and decrease even more for group E.
- After further analysis, groups A,B, and C were taken from people in regular mode using all their senses decreasing the predictability of their brain activity.

- Group B measurements were taken from people with eyes closed, therefore canceling the sight decrease the brain activity since there are less information to process.
- For group E even though a seizure is a chaotic state, it blocks most of the brain functions make it more predictable despite its chaotic nature.
- Our results were characterizing the mentioned relationships, thus a reproduction of the results in [1] is not possible since the nature of the data used will never show those results using PE as a comparing metric.

The method worked the way it was supposed to work but to get the desired results for any time series comparison we need to change the comparing metric.

2.3 Alternative Metric

After realizing that using PE was not giving the desired results it was time to reconsider the metric used for comparison. Instead of discarding everything done so far, the idea came up that maybe valuable underlying statistics information built during the permutation analysis was getting lost when it was all summarized into a number. There was the possibility that two series sharing the same value of PE, but one having a regularly appearance of every ranked permutation while the other had a clearly dominant one, however at the end the PE value was the same for both of them. Therefore a metric should be selected allowing us to measure differences among the underlying distributions leading to the final value of PE. For that reason, it was decided to try to use the Total Variation Distance (TVD).

2.3.1 Definition of Total Variation Distance (TVD)

The total variation distance is a distance measure that calculates the difference between two probability distributions. The total variation distance between two probability measures μ and υ on $\mathbb{R}$ is defined as [24]:

$$TVD(\mu, \upsilon) = \frac{1}{2} * \sum_{A=1}^{n} \|\mu(A) - \upsilon(A)\|$$

where μ and υ are probabilistic measurements of a random variable.

The confidence to use this new metric was built observing the results in [25] where the TVD approach was used to differentiate between Hidden Markov Models (HMM). This approach is relevant because our intermediate statistics form Markov Chains that are less complex than the HMM's. Therefore, the TVD approach theoretically is able to differentiate

between our generated statistics between any pair of time series. However, since the process of building statics from the ranked permutation create a probability distribution in the form of Markov chain, the formula to calculate TVD have to be modified.

In order to calculate the TVD metric to fit our data, we have to derive a joint version of the TVD definition. In our case, lets have two time series $X = (x_1, x_2, ..., x_k)$ and $Y = (y_1, y_2, ..., y_k)$, that are analyzed using the permutation entropy approach with window size n and shifting parameter $\tau = 1$. The possible amount of permutations is $n!$. For both time series, the initial distribution of the all their possible permutations will be given by a stationary distribution array of dimension 1 x $n!$, and the distribution of all the possible transitions between permutations by Transition Probability Matrix (TPM) of dimensions $n!$ x $n!$. So, for time series X we will have $Stat_x$ and TPM_x, and for time series Y we will have $Stat_y$ and TPM_y. Therefore the TVD value will be calculated as follows:

$$joint\_TVD(X,Y) = \frac{1}{2} * \sum_{i=1}^{n!} \sum_{j=1}^{n!} \|(Stat_x(i) * TPM_x(i,j)) - Stat_y(i) * TPM_y(i,j))\|$$

2.3.2 Reproduction Using TVD

Now that there was a new metric defined it was time to use it in the reproduced algorithm. Intuitively, the results should show better relationships between the groups this time. The method and the decision making process was not drastically changed so the null hypothesis was kept the same. The adjustments made to the original algorithm are presented in Appendix A. Only comparing setA with itself gave the following result:

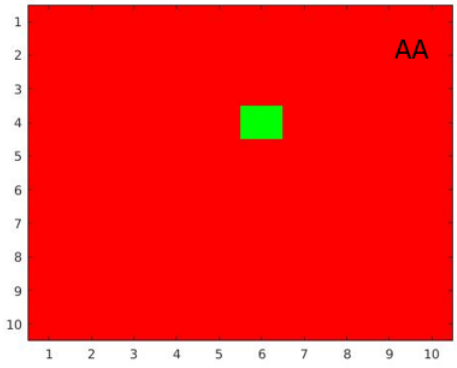

Figure 2.8: SetA with itself using TVD

Any other comparison gave all negative results. At this point obviously something was not being considered, so in order to identify where was the error further analysis was made.

2.3.3 TVD Histograms

Once again the results using the reproduced algorithm was not giving he expected results. So, the same way it was done with the Permutation Entropy value, an analysis of the TVD values was performed using this metric to directly perform all the possible comparisons between the EEG data sets. The TVD was calculated for all possible pairs and instead of being treated further they were kept and plotted in histograms, giving as result:

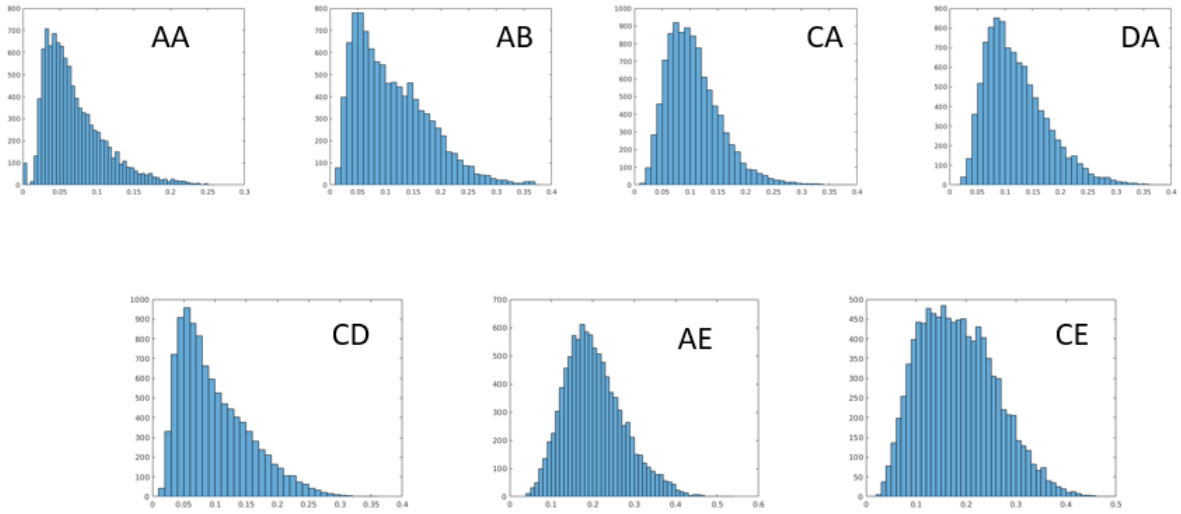

Figure 2.9: TVD Histograms

Surprisingly these results showed that:

- Comparison between groups A and B, and C and D presented similar graphics showing a small TVD value.
- Group E presented high values of TVD when compared with any other group.
- In any comparison between a group of healthy people with a group of epilepsy patients the values are higher than the ones presented lines above.
- Using TVD directly in our data it can be seen some of the anticipated relationships taking place.
- When we tried to used this approach with the bootstrap method we used the null hypothesis of TVD = 0, which was not able to show the results from the presented histograms.
- In order to use TVD as a decision metric for the proposed classification method instead of calculate an exact value null hypothesis, it should be an interval.

- A correct value ϵ must be found to be used as a threshold, such that the decision rule will be given by $0 =< TVD <= \epsilon$.

2.4 Section Conclusions

- The results presented in the paper using the comparison method based on PE is not reproducible and not accurate.
- The presented algorithm in [1] need to be changed to meet the classification metric parameters, instead of proposing an exact value null hypothesis, an interval probably might improve its performance.
- However, the permutation analysis do characterize the desired associations when the permutation statistics are used with the TVD metric.
- New algorithm need to be developed to characterize the relationships seen in the TVD histograms.
- The reproduced algorithm tried to determine a similarity relationship between time series, nevertheless we believe the new algorithm taking advantage of the TVD metric can go beyond and perform a classification job.

Chapter 3

Alternative Algorithms Using the TVD Distance

As seen in the previous chapter, the algorithm presented in [1] using permutation entropy was not able to characterize the difference between the different EEG data groups the way it was expected. Using the same approach, TVD distance was used as the new metric instead of PE leading to even worst differentiation. However, when the direct TVD distances were computed a differentiation was possible by observing the behavior of its histograms. Therefore, this metric was characterizing the behavior we were expecting from our data but the method used was not capturing that pattern. For that reason, a new algorithm has to be developed to take advantage of the TVD metric.

In this chapter, three algorithms are proposed to use the TVD metric along with the permutation analysis to not only state similarity relationships between time series but to classify them according to some given scenarios. The new methods will be defined and then some experimental results will be shown. The goal is to observe the performance of the three approaches to determine if this metric is efficient for classification purposes.

3.1 Algorithm Definition

The algorithms proposed assumed there are enough data sets to be used as training data for every group of sets that will be part of an scenario. The permutation analysis will be performed on this training data to build some underlying statistic regarding the scenarios to be classified. The algorithms will be explained, more detailed information of the generated algorithm code will be provided in Appendix A.

3.1.1 Average of TPM's

In the first place, for the first algorithm we build relative relationship. For example, we look at the distance between a point with respect to all others, whose relative position is defined in terms of a vector of finite length and a defined number of cluster. Let us assume each cluster has M distributions learned from M different time-series, and that the total number of cluster is N.

- For the training data in each group, the permutation analysis is performed for each time series.
- We extract the resulting TPM of each dataset of all the groups.
- Then, the TPM's are averaged within the same group, resulting in a single averaged TPM per group.
- A stationary distribution is calculated for every averaged matrix.
- With a TPM and stationary distribution per group, TVD is calculated between all groups.
- Therefore, an average TVD value was calculated for each group comparison.
- Having all the comparisson values, arrays were created anticipating what the TVD should be if a time series belongs to a specific group. For example if there are N groups $(G_1, G_2....G_N)$ and we assumed the new time series is from G_1 then an array called IfG_1 will be created containing the comparisson values of G_1 against the others in order $IfG_1 = [G_1G_1, G_1G_2....G_1G_N]$. This arrays will be called If-arrays in the following lines.
- Then we used our test data.
- For a new time series X the permutation analysis was performed calculating TPM and stat dist for it.
- Using those statistics, we calculated TVD comparisons between X and the averaged TPM's of each group, generating an array IfX containing the comparison values of the new sequence. For instance, taking again the case of a three group case $IfX = [XG_1, XG_2...XG_N]$.
- That array was compared with the previous calculated If-arrays using the L2 norm.
- The lowest value of L2 norm was determined to classify which group the test time series belongs to.

3.1.2 Average of TVD

This approach is similar from the previous one. The classification method is the same with the difference that instead of finding average TPM for each group to calculate TVD comparisons, now we calculated TPM for each time series of each group and found all the possible TVD comparisons and averaged those TVD values. The steps of this algorithm are:

- For the training data in each group, the permutation analysis was performed for each time series providing its TPM.

- With those TPM's we found all possible TVD comparisons between each group. For example if there was M training time series in group A and B, all possible MxM TVD values were calculated and stored in a long array. The same was done for all the possible group comparison including between the same group.
- Then, an average TVD value for each group comparison was calculated.
- With those average TVD comparissons, If-arrays were were defined the same way as in the previous algorithm.
- Then we used our test data, for each one of the testing time series the permutation analysis was performed calculating TPM and stationary distribution.
- Using those statistics, we calculated TVD comparisons for one testing time series against all the training sequences resulting in M TVD values for each group.
- Then, those TVD values are averaged for each group creating the array $IfX = [XG_1XG_2...XG_N]$
- That array was compared with the previously defined If-arrays using the L2 norm.
- The lowest value of L2 norm was determined to classify which group the test time series belongs to.

3.1.3 Average of TVD Using the Frobenius Norm

The third proposed approach is similar to the previous one, however instead of generating a single values of average TVD between all groups now a vector is generated. The procedure goes as follows:

The idea here is to calculate each group's relationship with the rest of N-1 ones, as well with itself is featured in a M by N matrix, where M is the number of labeled time series per group. The relative location of a group with respect to either itself or another group is defined by a vector of TVD of length M, which is attained through averaging out of M relative vectors in total.

- For the training data in each group, the permutation analysis was performed for each time series providing its TPM.
- With those TPM's we found all possible TVD comparisons between each group. For example if there was M training time series in G_1 and G_2, all possible MxM TVD values were calculated and stored in a matrix. This matrix will have in each row the comparison of one sequence of G_1 with all the sequences of G_2, resulting that each row will have M values.

- Then, for each group comparison matrix every row was averaged. Thus, a 1xM vector will result for each group comparison.
- Note that in this case G_1G_2 comparison is not the same as G_2G_1.
- With those average TVD comparisons vectors, If-arrays were defined the same way as in the previous algorithm. However this time we will not have a 1xN array but a NxM vector.
- Then we used our test data, for each one of the testing time series the permutation analysis was performed calculating TPM and stationary distribution.
- Using those statistics, we calculated TVD comparisons for one testing time series against all the training sequences resulting in M TVD values for each group, and store in a MxM matrix.
- Then, those TVD matrices are averaged row-wise for each group creating the NxM matrix IfX
- That matrix was compared with the previously defined If-arrays using the Frobenius Norm. These If-arrays now defined by a matrix, which reflects relative positions of its own sequences with respect to all groups including itself.
- The lowest value of Frobenius Norm was determined to classify which group the test time series belongs to.

3.2 Testing Using Previous EEG Data

To test the performance of the proposed algorithms some experiments were performed. For the first one, as a logical step, the same five groups of EEG data seen in previous chapters were used. Since each group have 100 time series, 80 of them were used for training purpose and 20 for testing. The majority of the time series were used for training in order to build a more reliable statistic. The three algorithms were tested using this data and the results are shown in Table 3.1. The results showed how many time series were correctly classified among the 20 time series tested for each group.

This results were encouraging since they maintain a relationship with the histograms presented in section X. Relevant observations and analysis from the results are:

- The results showed that for all three algorithms Group E is the easiest to classify, Groups A and B can be distinguish around 50% of the times, and Groups C and D are the most difficult to distinguish.
- The results maintain relationship with the histograms previously calculated for this data sets. In those, it was clear that the shift of the histogram for Group E was really

Table 3.1: Results using EEG data

Group	Avg. of TPM	Avg. of TVD	Avg. using Frob. Norm
A	50% (10)	55% (11)	50% (10)
B	40% (8)	40% (8)	35% (7)
C	20% (4)	20% (4)	35% (7)
D	25% (5)	25% (5)	30% (6)
E	85% (17)	80% (16)	75% (15)

pronounced from the rest while Groups C and D were mostly overlapping between each other.

- The algorithm using the Frobenius Norm present a better detection for Groups C and D than the other approaches.
- Despite the data being encouraging, the number of test data is so small that it cannot be considered as statistically representative to have decisive conclusions.
- More testing having a more representative amount of test data needed to be done.

3.3 Testing Using New EEG Data

Since the groups of EEG data sets used so far only contains 100 time series, new data sets were utilized. For a second algorithm testing, we utilized EEG data found in [26], where we have 7500 EEG time series coming from 5 epileptic patients with 4092 points each. Those time series are divided in two groups, Focal (F) and non-Focal (NF) each with 3750 time series. Focal EEG measurements comes from the parts of the brains involved during a seizure event while non focal measurements comes from brain areas that were not involved at seizure onset [27]. The goal was to test how well the algorithms can distinguish between those two groups. In order to do the testing, 400 time series were used as training data and 350 as testing data for each group.

In Table 3.2 the results are presented with a surprising outcome. Specifically for the focal group the detection percentage never goes beyond 50%, in other words tossing a coin would give a better detection than using the algorithms.

However, before reaching conclusions just with the given results more analysis was required. Similar to the testing trying to replicate the work in [1], the fact that the data used are EEG measurements and they require special attention since the brain operation is really complex. Thus, once more the TVD distances were calculated and graphed as histograms to observe the behavior of this metric. To do so, the data was separated according the way it was compressed. When downloaded the 3750 time series of each group were extracted

Table 3.2: Results using Focal and Non-Focal EEG data

Group	Avg. of TPM	Avg. of TVD	Avg. using Frob. Norm
Focal	46.57% (46.47%)	34% (119)	40.57% (142)
Non - Focal	52% (182)	66.28% (232)	56.28% (197)

from compressed folders containing 750 samples each. Intuition pointed that maybe there was some reason behind that way of organization so it was respected when calculating the distances. Therefore, the data was divided in 5 subgroups for each group, leading to Subgroups A to E having 750 time series each from the Focal Group, and Subgroups F to J from the Non-Focal Group. The TVD comparisons were performed between all time series in each subgroup, giving 562500 TVD values per comparison that were graphed into histograms.

Some examples of the resulting histograms are showed in Figure 3.1, and there was no necessity to show a more considerable amount of histograms of all the 54 possible comparisons between the subgroups because the the resulting graphics are almost identical. With this pattern of the behavior of the TVD metric is clear that the problem is not related to the developed algorithms but more it is linked that the nature of the data do not allow the TVD to find differences between the time series.

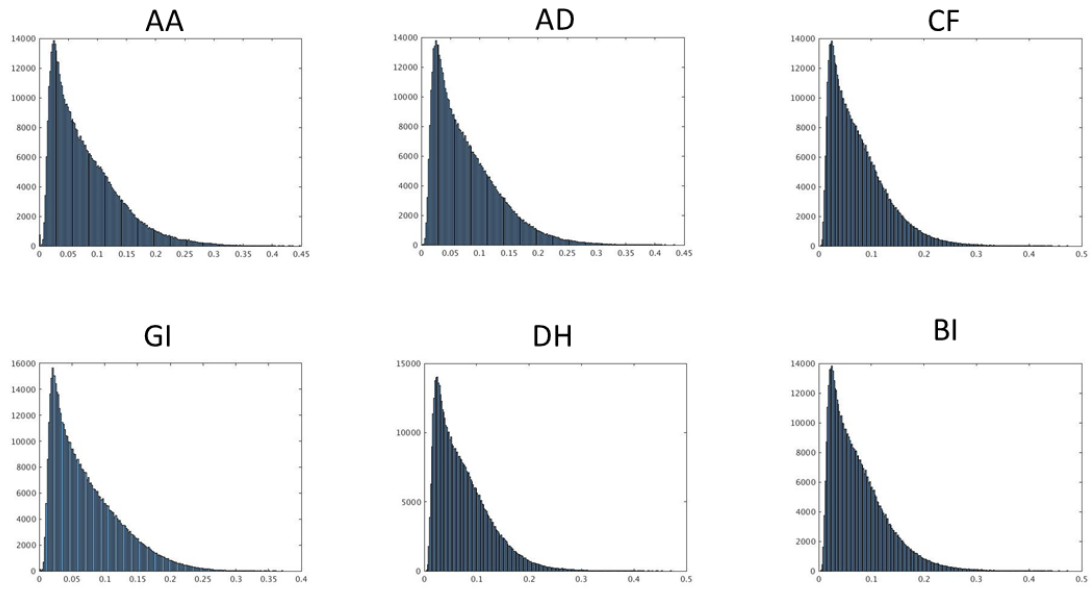

Figure 3.1: Focal and Non-Focal EEG data sets TVD Histograms

EEG data has proven to be too complex and unpredictable to test the proposed algorithms in a conclusive way. More testing was required with data able to be measured and that was able to have a more objective way to benchmark.

3.4 Testing Using Self Generated Data

After analyzing the results obtained in the previous section, the decision made was to test the proposed algorithms using self generated data so the anticipated results could be more predictable. The goal is to create four groups with statistical distributions that are easy to differentiate, and to be able to have a proved method of differentiation as benchmark to be compared with the performance of the proposed algorithms.

The self generated data consisted in four groups containing 20000 time series each with a length of 4000 elements. Each group was generated as follows:

- Group A: $[X_1\ X_2 ....\ X_n]$ i.i.d time series following standard normal distribution, i.e. $\mathcal{N}(0,1)$.
- Group B: Filtered version of Group A using the equation $Y_n = \alpha_1 * X_n + \alpha_2 * X_{n-1}$
- Group C: Filtered version of Group A using the equation $Y_n = \alpha_1 * X_n + \alpha_2 * Y_{n-1}$.
- Group D: Filtered version of Group A using the equation $Y_n = \alpha_1 * X_n + \alpha_2 * X_{n-1} + \beta * Y_{n-1}$.

The resulting filtered versions of Group A will also be Gaussian therefore the Maximum Likelihood (ML) approach can be used as benchmark method. To do so, we have to assume the time series to be analyzed belongs to one of the the defined groups

$$\begin{pmatrix} Y_1 \\ Y_2 \\ . \\ . \\ . \\ Y_k \end{pmatrix} \sim H_j, j \in \{1,2,3,4\}$$

Then, we calculate the log-probability of that assumption $log(P[Y_1, Y_2 ... Y_k | H_j])$, where that probability will be calculated using the multivariate Gaussian density function:

$$f_Y(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} exp(-0.5(x-\mu)^T)\Sigma^{-1}(x-\mu))$$

The only unknown to calculate the wanted probability for each time series will be the co-variance matrix, therefore we approximate that matrix using;

$$Cov(Y) = \frac{1}{n}\sum_{j=1}^{n}(Y_i.Y_i^T)$$

Thus, each time series will be evaluated assuming the four different groups, and the assumption giving the maximum log-probability will be selected to classify the given time series.

Now, all the steps are defined to start testing the algorithms and the ML approach with the generated data. However, to have even more control of the predicted outcome before we start the testing, the behavior of the statistics according the filtering coefficients have to be calculated. To do so, we would like to choose appropriate parameters α and β to control the differentiability between these Gaussian distributions by resorting to matrix norms of their resulting covariance matrices. Using the Frobenius Norm, the covariance matrices of the four groups were compared giving a clear view of how the difference between those matrices vary according the selected coefficients. Therefore, we are able to know which coefficients will lead to most distinguishable groups and which ones will shrink that difference.

Each time series has 4000 elements, complicating the calculation of the co-variance matrix since the inverse of a 4000x4000 matrix would be necessary. For that reason, for the calculations involving co-variance matrices only the last 100 elements of each time series were used. In Table 3.3 the variation between co-variance matrices distances according the chosen coefficients is presented. The results obtained were used in the election of the coefficients in the different tests performed.

3.4.1 Test1

For the first test the idea was to select the coefficients that generate the most distinct groups. Thus, according to Table 3.3 the combination that give the desired result is $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$. The first set of results in Table 3.4 correspond to the test using the three proposed algorithms and the ML approach. To build the results, 600 time series per group were used as training data and 400 as testing data. In this case, we used the full length (Length = 4000) of the time series when running the proposed algorithms and only the last 100 elements when running the ML approach. In the table results the variables like CountA, represent the counting of the time series that were correctly classified according the group that is being analyzed.

Table 3.3: Co-variance Distances Variation According to the Filtering Coefficients

	α_0	α_1	β	AB	AC	AD	BC	BD	CD
1	0.7	0.3	0.4	5.1506	5.2071	8.7636	1.0746	6.8308	7.2368
2	0.6	0.4	0.4	5.9124	6.3244	9.0762	1.6969	6.8191	7.5621
3	0.5	0.5	0.4	6.1413	7.2045	9.1288	2.4396	6.7693	8.022
4	0.4	0.6	0.4	5.8949	6.3063	9.0351	1.6948	6.7814	7.5276
5	0.3	0.7	0.4	5.1975	8.6127	8.8067	4.5524	6.8448	9.71
6	0.2	0.8	0.4	3.9974	9.149	8.4085	6.0216	6.9512	10.8946
7	0.1	0.9	0.4	2.3807	9.5916	7.9661	7.812	7.1627	12.2904
8	0.2	0.8	0.3	3.9974	9.149	6.3729	6.0216	4.3959	8.8279
9	0.2	0.8	0.5	3.9974	9.149	11.8135	6.0216	10.7705	14.2096
10	0.2	0.8	0.6	3.9974	9.149	17.5758	6.0216	16.8707	19.7456
11	0.2	0.8	0.7	3.9974	9.149	28.4433	6.0216	28.0033	30.3089
12	0.2	0.8	0.8	3.9974	9.149	54.0884	6.0216	53.8558	55.4708
13	0.2	0.8	0.9	3.9974	9.149	155.1702	6.0216	155.0914	155.9071

Table 3.4: Full length Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	100%	100%	100%
Avg TVD	Full Length	100%	100%	100%	100%
Frob. Norm	Full Length	100%	100%	100%	100%
ML	Last 100	100%	100%	100%	100%

Every single method used was able to classify perfectly the test data according the group it belongs to. Now, this results were obtained using the full length of the sequences that is 4000 elements. Not always the sequences to be analyzed will posses this extensive amount of elements. Therefore, to test the performance when the sequences are shorter the test and training were trimmed to only a constraint number of elements. The number of computed time series are the same (600 as training data and 400 as testing), but the length of those sequences will be trimmed to 100, 200, 300, 400, and 500. In the following tables results all testing data was constraint according to the case, and for the training both the full length and trimmed versions were used to observe how it affects the results.

In Table 3.5 the results for only using the last 100 elements for testing data are presented. As expected, the classification rate success diminished for all groups being more dramatically

for Group C. Next, the result tables for the cases for the last 200, 300, 400, and 500 elements are presented. It is necessary to point that for the next cases we also increased the elements used for the ML approach calculations.

Table 3.5: Last 100 Elements Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	95.25% (381)	90% (360)	53.5% (214)	99.25% (397)
	Last 100	95% (380)	89.25% (357)	54% (216)	99.25% (397)
Avg TVD	Full Length	92.5% (382)	88.5% (354)	51.75% (207)	99.25% (397)
	Last 100	82.75% (331)	84% (336)	72% (288)	96.25% (385)
Frob. Norm	Full Length	92.5% (382)	89% (356)	51.75% (207)	99.25% (397)
	Last 100	82% (328)	84.75% (339)	72.5% (290)	96% (384)
ML		100%	100%	100%	100%

Table 3.6: Last 200 Elements Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	96.75% (387)	99% (396)	83% (332)	100%
	Last 200	96.5% (386)	99% (396)	84% (336)	100%
Avg TVD	Full Length	97% (388)	98.75% (395)	81.25% (325)	100%
	Last 200	92.5% (370)	92.75% (383)	87.25% (349)	100%
Frob. Norm	Full Length	97% (388)	98.75% (395)	80.75% (323)	100%
	Last 200	92.25% (369)	97% (388)	87% (348)	100%
ML	Last 200	100%	100%	100%	100%

Table 3.7: Last 300 Elements Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	98.25% (393)	99.5% (398)	92.25% (369)	100%
	Last 300	98% (392)	99.5% (398)	92.5% (370)	100%
Avg TVD	Full Length	98.5% (394)	99.5% (398)	92% (368)	100%
	Last 300	97.75% (391)	98.75% (395)	93.25% (373)	100%
Frob. Norm	Full Length	98.5% (394)	99.5% (398)	92% (368)	100%
	Last 300	97.75% (391)	99% (396)	93.5% (374)	100%
ML	Last 300	100%	100%	100%	100%

Table 3.8: Last 400 Elements Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.25% (397)	99.5% (398)	98.25% (393)	100%
	Last 400	99.25% (397)	99.5% (398)	98.25% (393)	100%
Avg TVD	Full Length	99.25% (397)	99.5% (398)	98.25% (393)	100%
	Last 400	97.75% (391)	99.25% (397)	98.25% (393)	100%
Frob. Norm	Full Length	99.25% (397)	99.5% (398)	98.25% (393)	100%
	Last 400	97.75% (391)	99.5% (398)	98.25% (393)	100%
ML	Last 400	100%	100%	100%	100%

Table 3.9: Last 500 Elements Test Data Results with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta = 0.9$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.25% (397)	99.75% (399)	98.25% (393)	100%
	Last 500	99% (396)	99.75% (399)	98% (392)	100%
Avg TVD	Full Length	99.25% (397)	99.75% (399)	97.75% (391)	100%
	Last 500	98.5% (394)	99.5% (398)	98% (392)	100%
Frob. Norm	Full Length	99.25% (397)	99.75% (399)	97.75% (391)	100%
	Last 500	98.5% (394)	99.5% (398)	98% (392)	100%
ML	Last 500	100%	100%	100%	100%

As seen in Tables 3.6, 3.7, 3.8, and 3.9, the classification success rate increased when the analysis were made with longer sequences. It is interesting to observe that when the length was 300 elements the classification was over 90% for all approaches and all groups. Interesting remarks are:

- Group C suffered the most in this case when the length got really short, with a performance around between 50% and 60%.
- Group D was almost perfectly classified for all methods with any length.
- As thought, a length of 100 affected all three algorithms success rate.
- Contrary to first intuition, using shorter length or full length for the training data did not have a great impact in the final performance. Except when the length was 100, where using a short length for training data made a difference for group C in the Avg. of TVS and Frob Norm approaches. The performance with regular training sequences was around 50% while the shorter version resulted in success around 70 %.
- Overall, the three methods had a similar behavior.
- It was encouraging to see that with a short length of 300 the classification was already above 90% for all cases.

3.4.2 Test2

Previously the best case for classification purposes was presented. Now, it is required to test the algorithms with a more difficult data sets to classify. Using the knowledge of the behavior of the statistics according to the selected coefficients, the second testing was done using the coefficients $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$. The procedure for the testing was the same as in the previous section. The results for the full length time series were:

Table 3.10: Full length Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	100%	100%	100%
Avg TVD	Full Length	100%	100%	100%	100%
Frob. Norm	Full Length	100%	100%	100%	100%
ML	Last 100	97.5% (390)	98.25%(393)	100%	99.75% (399)

Again the classification was perfect when using the full length for training and testing data using the proposed algorithms, but it was not perfect using the ML approach. Once more, the results using only the last 100 elements of the sequence were calculated:

Table 3.11: Last 100 Elements Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	90.25% (361)	33% (132)	19.75% (79)	85% (340)
	Last 100	89.5% (358)	34.5% (139)	17% (68)	87% (348)
Avg TVD	Full Length	91.75% (367)	31.5% (126)	17% (68)	86.25% (345)
	Last 100	65.75% (263)	66% (264)	83% (332)	7% (28)
Frob. Norm	Full Length	93.75% (375)	32.5% (130)	21.25% (85)	86.5% (346)
	Last 100	66.5% (266)	72.75% (291)	75% (300)	0%
ML	Last 100	97.5% (390)	98.25%(393)	100%	99.75% (399)

This time the classification success rate drastically dropped for groups B and C. By simple observation looks like using the shorter length version of the sequences for training give slightly better results in this case. Next, the result tables for the cases for the last 200, 300, 400, and 500 elements are presented.

Table 3.12: Last 200 Elements Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	91.75% (367)	69.75% (279)	48.75% (195)	84% (336)
	Last 200	91.25% (365)	71.25% (285)	47.75% (191)	84.25% (337)
Avg TVD	Full Length	92.25% (369)	67.5% (270)	44.75% (179)	83% (332)
	Last 200	77% (308)	85.5% (342)	64.25% (257)	67% (268)
Frob. Norm	Full Length	92.25% (369)	67.5% (270)	44.75% (179)	83% (332)
	Last 200	75% (300)	87% (348)	65% (260)	65.75% (263)
ML	Last 200	99.75% (399)	99.5% (398)	100%	100%

Table 3.13: Last 300 Elements Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	92.25% (369)	85% (340)	59.75% (239)	86.75% (347)
	Last 300	91.25% (365)	86% (344)	57.75% (231)	88% (352)
Avg TVD	Full Length	93.5% (374)	82.25% (329)	56.75% (227)	86.75% (347)
	Last 300	83.75% (335)	92.5% (370)	68.75% (275)	77.25% (309)
Frob. Norm	Full Length	93.5% (374)	82.25% (329)	56.75% (227)	86.75% (347)
	Last 300	81.5% (326)	93.5% (374)	67.75% (271)	76.25% (305)
ML	Last 300	100%	100%	100%	100%

Table 3.14: Last 400 Elements Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	95% (380)	90.75% (363)	65.5% (262)	87.25% (349)
	Last 400	95% (380)	90.5% (362)	64.5% (258)	87.5% (350)
Avg TVD	Full Length	95.5% (382)	90% (360)	63.5% (258)	86.5% (350)
	Last 400	89.25% (357)	93.25% (373)	71% (284)	78% (312)
Frob. Norm	Full Length	95.5% (382)	90% (360)	62% (252)	85.5% (350)
	Last 400	89% (356)	93.5% (374)	71.5% (286)	76.5% (306)
ML	Last 400	100%	100%	100%	100%

Table 3.15: Last 500 Elements Test Data Results with $\alpha_1 = 0.2$, $\alpha_2 = 0.8$, and $\beta = 0.6$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	97% (388)	93.5% (378)	75% (300)	90.75% (363)
	Last 500	97% (388)	94.5% (378)	73.5% (294)	91.25% (365)
Avg TVD	Full Length	97.5% (390)	94% (376)	71.75% (287)	89.5% (358)
	Last 500	94.75% (379)	96% (384)	74.5% (298)	83.25% (333)
Frob. Norm	Full Length	97.5% (390)	94% (376)	72% (288)	89.5% (358)
	Last 500	94.25% (377)	96.5% (386)	75% (300)	82.25% (329)
ML	Last 500	100%	100%	100%	100%

As seen in Tables 3.12, 3.13, 3.14, and 3.15, the classification success rate increased when the analysis were made with longer sequences. Nevertheless, this increment is not as fast as seen in Test 1. Group C in this case is in the 70% range of classification success rate. Eventually all Groups will reach perfect classification rate but for the selected coefficients it will be required a larger minimum sequence length to achieve it. Moreover, as sequence

length increases, we can hardly see significant difference of performance between different methods.

3.4.3 Test3

This test explored the case when the coefficients supposedly will generate harder to classify groups. The combination to build this scenario is given by $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$. The procedure for the testing was the same as in the previous section. The results for the full length time series were:

Table 3.16: Full length Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	100%	100%	100%
Avg TVD	Full Length	100%	100%	100%	100%
Frob. Norm	Full Length	100%	100%	100%	100%
ML	Last 100	99.75% (399)	77.25% (309)	77% (308)	93% (372)

The proposed algorithms were able to perfectly classify all time series despite the given scenario. However, in this case the ML approach dropped its performance specially for groups B and C. Of course, this is for an ML approach with only 100 elements per time series when the rest of the algorithms were using their full length making it an unfair comparison. The results for the test using only the last 100 elements for the test data are shown in Table 3.17, where the drop in the performance is clear. Surprisingly, this results showed that when the training data is also of short length the performance for groups B and C improved considerably when Average of TVD and Average of TVD with Frobenius Norm approaches were used.

As seen in Tables 3.18, 3.19, 3.20, and 3.21, the pattern of improvement while increasing the element number in the sequences is still present. It is interesting how the difference in performance is big between using the full or short length sequence for the training data, but that that difference became minimal when the length reached 400 points.

Table 3.17: Last 100 Elements Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	92.75% (371)	12.5% (50)	9% (36)	95.5% (382)
	Last 100	91.25% (365)	13% (52)	8.75% (35)	95.25% (381)
Avg TVD	Full Length	93% (372)	10.5% (42)	8.25% (33)	95.5% (382)
	Last 100	62.25% (249)	51.75% (207)	43.5% (174)	76% (304)
Frob. Norm	Full Length	93% (372)	10.75% (43)	8.25% (33)	95.5% (382)
	Last 100	49% (196)	57.5% (230)	53% (212)	64.5% (258)
ML	Last 100	99.75% (399)	77.25% (309)	77% (308)	93% (372)

Table 3.18: Last 200 Elements Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	93.75% (375)	45.25% (181)	37% (148)	96.5% (386)
	Last 200	94% (376)	45.25% (181)	38.25% (153)	96% (384)
Avg TVD	Full Length	94% (376)	41.25% (165)	38.5% (154)	96.25% (385)
	Last 200	75.25% (301)	67.25% (269)	64.5% (258)	80.75% (323)
Frob. Norm	Full Length	94% (376)	41.5% (166)	38.25% (153)	96.5% (386)
	Last 200	70.75% (283)	69.75% (279)	66.75% (267)	76.75% (307)
ML	Last 200	100%	91% (354)	92.5% (370)	96.75% (387)

Table 3.19: Last 300 Elements Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	93.75% (375)	45.25% (181)	37% (148)	96.5% (386)
	Last 300	94% (376)	45.25% (181)	38.25% (153)	96% (384)
Avg TVD	Full Length	94% (376)	41.25% (165)	38.5% (154)	96.25% (385)
	Last 300	75.25% (301)	67.25% (269)	64.5% (258)	80.75% (323)
Frob. Norm	Full Length	94% (376)	41.5% (166)	38.25% (153)	96.5% (386)
	Last 300	70.75% (283)	69.75% (279)	66.75% (267)	76.75% (307)
ML	Last 300	100%	91% (354)	92.5% (370)	96.75% (387)

Table 3.20: Last 400 Elements Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	96.75% (387)	75% (300)	70.75% (283)	97.5% (390)
	Last 400	96.5% (386)	74.75% (299)	70.25% (281)	97.5% (390)
Avg TVD	Full Length	96.5% (386)	74% (296)	71.5% (286)	97.25% (389)
	Last 400	89.25% (357)	84.25% (337)	80.5 (322)	93.25 (373)
Frob. Norm	Full Length	96.5% (386)	74.25% (297)	71.5% (286)	97.25% (389)
	Last 400	87.75% (351)	85% (340)	80.5% (322)	92.5% (370)
ML	Last 400	100%	97% (388)	97.75% (391)	99.75% (399)

Table 3.21: Last 500 Elements Test Data Results with $\alpha_1 = 0.7$, $\alpha_2 = 0.3$, and $\beta = 0.3$

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	97.5% (390)	84% (336)	77.75% (311)	98% (392)
	Last 500	97.5% (390)	83.25(333)	77.5% (310)	98.75% (395)
Avg TVD	Full Length	97.5% (390)	83.75% (335)	78.75% (315)	98% (392)
	Last 500	91.75% (367)	87.5% (350)	83.5% (334)	96.5% (386)
Frob. Norm	Full Length	97.5% (390)	84% (336)	78.25% (313)	98% (392)
	Last 500	91.5% (366)	88.75% (355)	83.75% (335)	95.75% (383)
ML	Last 500	100%	98.5% (394)	99% (396)	99.75% (399)

Some interesting remarks after observing the results of testing using shorter length sequences are:

- Avg of TVD and Frob. Norm approaches presented better performance when using short length sequence as training data for groups B and C.
- This difference got really short when the length was 500 though.
- Group D presented overall good detection rate overall. However in this case classification dropped when using short length sequence as training data for Avg of TVD and Frob. Norm approaches, this happened for lengths up to 300, for larger lengths the difference between using different length sequence as training data got minimal.

3.4.4 General Observations

After generating the results from the three tests using our own generated data, we can observe that:

- When used the full length of the sequences for training and testing data the algorithms were able to perform perfect classification no matter the coefficients used.
- Depending on the coefficients used, the classification suffered a severe drop in the success rates when the sequence's length is really short.

- No matter what coefficients are used to generate the data, it is estimated that with around 1000 elements in the sequence the classification using any of the three algorithms will be really close to 100% success rate.
- The performance among algorithms did not vary drastically in the test done. For sequences lengths of 100 or 200, the Frob. Norm approach presented a little better results when using also short length sequences as training data.
- The groups consisted in data sharing the same distribution, however real life data most likely will be the product of a mixture of distributions.

3.5 Testing Using Self Generated Data Under Mixture Distributions

It was proved that the proposed classification algorithms are able to perfectly classify the time series when the data in each group share the same distribution. Nevertheless, real data sets most of the times do not come from specific distributions but are a mixture of them. For instance, communications over the spectrum over a polluted air interface suffering to extensive exposure from different sources and types of noise. Since this multiple distribution type data is the one most likely to be found, it was necessary to carry on tests to measure this type of data. We used the same idea of creating self generated data to simulate one of those scenarios and test again the proposed algorithms. Once again, the ML approach will be used as a benchmark approach, however the statistic built from the data will not be completely Gaussian anymore it was considered interesting to check how this approach will behave in the new scenario. Intuitively we expect that the performance of the all classifying approaches will drop.

The method to build the mixed scenario is to create the filtered version of the data sets in Group A but using different filtering coefficients among the same group. For example, Group B uses the filtering equation $Y_n = \alpha_1 * X_n + \alpha_2 * X_{n-1}$, so instead of using the same α's and β for all the series in the group, we will use different coefficients to create the sequences. This will be divided in block, so if 4 different coefficients combination are used the total amount of time series will be divided by 4 and each block will have time series generated from a specific set of coefficients. Therefore, a block means a set of time series sequences following a particular distribution. Then, we will pick for the training data equal number of sequences from each block, for instance if there are 4 sets of coefficients then 4 blocks of data will be created and therefore 1/4 of the training data will come from each block. The same was done for the testing data. This was not applied for Group A since this group is the generator of the Gaussian sequences used to generate the rest of the groups. The testing carried on are presented next.

3.5.1 Mixed Test1

For the mixed test intended we used the same coefficients used in Sections 3.4.1, 3.4.2, and 3.4.3.Thus, we will have 3 blocks of different sets of coefficient given by:

- Block 1: $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta_1 = 0.9$.
- Block 2: $\alpha_3 = 0.2$, $\alpha_4 = 0.8$, and $\beta_2 = 0.6$.
- Block 3: $\alpha_5 = 0.7$, $\alpha_6 = 0.3$, and $\beta_3 = 0.3$.

For the training data part 900 sequences were used, selecting 300 of them from each block. For the testing part 600 time series were evaluated picking 200 of them from each block. Once this was done, the procedure was the same as in the previous section. The results of the the testing using the full length time series for training and testing data were:

Table 3.22: Full length Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	36% (216)	84.16% (505)	74.33% (446)
Avg TVD	Full Length	100%	15.5% (93)	74.83% (449)	70.5% (423)
Frob. Norm	Full Length	100%	26.5% (159)	95% (570)	37% (222)
ML	Last 100	100%	80.83% (485)	68.16% (409)	75.33% (452)

Table 3.22 shows interesting results for this new new scenario. Of course Group A has no change from previous evaluation since it has not changed. For the first time, the success rate is not perfect for the full length test. Group B suffered a huge drop of its classification efficiency for the proposed algorithms, but remain in high percentage for the ML approach. Contrary to this, Group C presented better performance in all the algorithms when compared with ML. For Group D the performance is similar for all methods except the one using the Frobenius Norm. Also, it is the first time that there is a prominent difference between the 3 algorithms. In this test, the approach using the Average of TPM's have a a better overall performance. According to past testing, this numbers should drop for all groups when the number of elements in each time series become small so lets check if this previous observation still holds.

Table 3.23 show the results when only the last 100 elements are used. Surprisingly, for Group D the success rate for some test is even better than using the full length of the time

series. In fact, increasing the number of elements in Tables 3.24, 3.25, 3.26, and 3.27 do not affect too much the classification performance of Group D and Group B, that with small lengths like 200 or 300 elements the results are really similar to the one using the full length. Only Group C kept the trend seen before of performance degrading when the length tends to be in the hundreds range.

Table 3.23: Last 100 Elements Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	96.66% (580)	25.5% (153)	16.33% (98)	82.16% (493)
	Last 100	96.83% (581)	23.83% (143)	16.83% (101)	82.5% (495)
Avg TVD	Full Length	91.16% (547)	34.66% (208)	1.5% (9)	86.5% (519)
	Last 100	41% (246)	76.5% (459)	7.8% (47)	54.16% (325)
Frob. Norm	Full Length	95.66% (574)	9.6% (58)	37.83% (227)	77.16% (463)
	Last 100	0%	98.5% (591)	0%	30.83% (185)
ML	Last 100	99.66% (598)	80.83% (485)	67.5% (405)	77.5% (465)

Table 3.24: Last 200 Elements Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	98.83% (593)	32% (192)	39.83% (239)	79.83% (479)
	Last 200	99% (594)	29.5% (177)	40.5% (243)	79.83% (479)
Avg TVD	Full Length	97.83% (587)	45.66% (274)	12% (72)	81.5% (489)
	Last 200	95% (570)	27.83% (167)	54.5 (327)	62.16% (373)
Frob. Norm	Full Length	99.33% (596)	7.33% (44)	65.16% (391)	65.5% (393)
	Last 200	97% (582)	5.33% (32)	83% (498)	41.66% (250)
ML	Last 200	100%	89.66% (538)	67% (402)	75% (450)

Table 3.25: Last 300 Elements Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.66% (598)	33.16% (199)	49.5% (297)	76.66% (460)
	Last 300	99.66% (598)	32.5% (195)	49.16% (295)	77.83% (467)
Avg TVD	Full Length	99.16% (595)	45.5% (273)	26.83% (161)	77% (462)
	Last 300	98.16% (589)	24.5% (147)	57% (342)	63.83% (383)
Frob. Norm	Full Length	99.83% (599)	10.5% (63)	73.66% (442)	58.33% (350)
	Last 300	99% (594)	1% (6)	85.33% (512)	40.66% (244)
ML	Last 300	100%	93.66% (562)	66.83% (401)	75% (450)

Table 3.26: Last 400 Elements Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.83% (599)	34.5% (207)	58% (348)	76% (456)
	Last 400	99.83% (599)	33.83% (203)	58% (348)	77.83% (464)
Avg TVD	Full Length	99.5% (597)	36.66% (220)	38.66% (232)	75.33% (452)
	Last 400	99% (594)	22.16% (133)	58.16% (349)	64.33% (386)
Frob. Norm	Full Length	100%	10.83% (65)	78.5% (471)	54.66% (328)
	Last 400	99.66% (598)	1% (6)	86% (516)	38.66% (232)
ML	Last 400	100%	96.16% (577)	66.66% (400)	76% (456)

Table 3.27: Last 500 Elements Test Data Results for Mixed Test 1

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	34.16% (205)	62.5% (375)	75.83% (455)
	Last 500	100%	33.83% (203)	62% (372)	76.5% (459)
Avg TVD	Full Length	100%	34% (204)	48.66% (292)	74.16% (445)
	Last 500	100%	21.16% (130)	60.5% (363)	65% (390)
Frob. Norm	Full Length	100%	10.33% (62)	81.33% (488)	53.33% (320)
	Last 500	100%	0.83% (5)	86% (516)	38% (228)
ML	Last 500	100%	97.33% (584)	66.66% (400)	75.33% (452)

The results showed a less intuitive behavior of the algorithms performance. Instead of an improvement in the performance while increasing the length of the sequences, the success rate presented a more stationary behavior with some fluctuations. Some important remarks are:

- For the Frob. Norm approach when the length of testing and training data was 100 the success rate for group C was 0% and for group B was almost perfect. However those results were inverted for length 200 and the rest of lengths. Is still unknown why this drastic change of behavior occurred.
- There is no clear trend whether using full or short length for the time series improves the algorithm performance.
- Overall, Avg. of TPM approach had the better results, only falling short in its classification for group C against the Frob. Norm approach. However, this last method had not so good performance for group B and D.

3.5.2 Mixed Test2

The previous test used 3 blocks of coefficients to simulate a mixed distribution approach. Intuition lead to think that the performance will drop when adding another level of distribution mixture. For this new testing, 4 blocks were used, with different sets of coefficient given by:

- Block 1: $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, and $\beta_1 = 0.9$.
- Block 2: $\alpha_3 = 0.2$, $\alpha_4 = 0.8$, and $\beta_2 = 0.6$.
- Block 3: $\alpha_5 = 0.7$, $\alpha_6 = 0.3$, and $\beta_3 = 0.3$.
- Block 4: $\alpha_7 = 0.1$, $\alpha_8 = 0.2$, and $\beta_4 = 0.1$.

The first three sets of coefficients remained the same from Test 1 and the additional set of coefficients were selected to try to make the classification even harder. In this case, for the training data part 1200 sequences were used, selecting 300 of them from each block. For the testing part 800 time series were evaluated picking 200 of them from each block. In Table 3.28 the results of the classification under the new scheme is shown.

Table 3.28: Full length Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	100%	48.37% (387)	58.62% (469)	71.13% (569)
Avg TVD	Full Length	100%	48.75% (390)	63.38% (507)	61.25% (490)
Frob. Norm	Full Length	100%	74.13% (593)	48.38% (387)	28.5% (228)
ML	Last 100	100%	55% (440)	70.88% (567)	79.16% (475)

It is interesting to note that the overall success rate for Group B increased for the three proposed algorithms but decreased for the ML approach. Group D also was better classified for all approaches but Group C got opposites results.

Table 3.29: Last 100 Elements Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	96.38% (771)	20.38% (163)	14% (112)	81.75% (654)
	Last 100	96.25% (770)	18.5% (148)	14.63% (117)	82.63% (661)
Avg TVD	Full Length	87% (696)	15.88% (127)	17.63% (141)	88.38% (707)
	Last 100	79.25% (634)	37.63% (301)	41.63% (333)	55.13% (4410
Frob. Norm	Full Length	95.5% (764)	30.38% (243)	11.75% (94)	73% (584)
	Last 100	82.63% (659)	48.25% (386)	43.5% (340)	31.75% (254)
ML	Last 100	100%	55% (440)	70.88% (567)	59.38% (475)

Table 3.30: Last 200 Elements Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	97% (776)	33.75% (270)	29.75% (238)	75.63% (605)
	Last 200	97% (776)	33.25% (266)	30.38% (243)	75.5% (604)
Avg TVD	Full Length	93.63% (749)	31.25% (250)	32.88% (263)	77.38% (619)
	Last 200	89.75% (718)	37.63% (301)	46.13% (369)	55.38% (443)
Frob. Norm	Full Length	97.13% (777)	44.38% (355)	24.38% (195)	55.13% (441)
	Last 200	94.5% (756)	46.75% (374)	46.38% (371)	28.88% (231)
ML	Last 200	100%	56% (448)	73% (584)	60.38% (483)

Table 3.31: Last 300 Elements Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	98.75% (790)	37.75% (302)	39.63% (317)	74.75% (598)
	Last 300	99% (792)	37.25% (298)	40% (320)	74.88% (599)
Avg TVD	Full Length	97.38% (779)	36.5% (292)	41.75% (334)	74.13% (593)
	Last 300	95.63% (765)	37.63% (301)	48.38% (387)	53.5% (428)
Frob. Norm	Full Length	99.5% (796)	48.38% (387)	31.25% (250)	45.13% (361)
	Last 300	98.75% (790)	49.63% (397)	45.13% (361)	28% (224)
ML	Last 300	100%	55.88% (447)	73.5% (588)	61% (488)

Table 3.32: Last 400 Elements Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.88% (799)	41.38% (331)	45.38% (363)	72.75% (582)
	Last 400	99.88% (799)	40.88% (327)	45.38% (363)	73.13% (585)
Avg TVD	Full Length	98.88% (791)	41.38% (331)	49.38% (395)	69.88% (559)
	Last 400	98.63% (789)	41% (328)	53.38% (427)	53.13% (449)
Frob. Norm	Full Length	100%	55.13% (441)	35.38% (283)	38% (304)
	Last 400	100%	56.5% (452)	41.75% (334)	28.88% (231)
ML	Last 400	100%	57.5% (460)	74.5% (596)	60.38% (483)

Table 3.33: Last 500 Elements Test Data Results for Mixed Test 2

		Count A	Count B	Count C	Count D
Avg Matrix	Full Length	99.88% (799)	41.38% (331)	45.38% (363)	72.75% (582)
	Last 500	99.88% (799)	40.88% (327)	45.38% (363)	73.13% (585)
Avg TVD	Full Length	98.88% (791)	41.38% (331)	49.38% (395)	69.88% (559)
	Last 500	98.63% (789)	41% (328)	53.38% (427)	53.13% (449)
Frob. Norm	Full Length	100%	55.13% (441)	35.38% (283)	38% (304)
	Last 500	100%	56.5% (452)	41.75% (334)	28.88% (231)
ML	Last 500	100%	57.5% (460)	74.5% (596)	60.38% (483)

Observing the all the result table is noted that this time the rule of performance decrease when the length of the sequence tends to be really small. As contrary as expected, the overall performance for four blocks was not worse and sometimes even better than the approach with three blocks. Another interesting remark is that the ML approach did not significantly increase its performance when the length of the time series was increased. In this case is harder to point one approach as being than rest because each one performed better for a specific group.

3.5.3 General Observations

After running all the experiments some major remarks that results showed are:

- For non-parametric approaches the results are not always intuitive.
- Using the Average of TPM's approach gave us more consistent results for the mixed testing, something that was not clear for single distributed groups where the algorithms performed in really similar ways.
- Adding an extra layer of mixture contrary to what was initially thought, gave better classification performance.
- Despite that ML approach usually gave consistent good results for the experiments, its computation cost since involve calculating inverse of matrices is very high. The proposed algorithms are computed in a less complex way.

- Also, when using even more complex mixed distributed data sets the ML approach will be useless since it is assuming an underlying Gaussian distribution. The proposed algorithms will still work for this kind of data sets since they do not assume anything from the incoming data.

- All the results presented were considering a permutation window of n = 3. Some other testing was performed increasing that window but it did not have a significant impact in the final results.

- Even when results were not as optimal as wished, the proposed algorithms are promising as a first step in order to classify time series in a non-para-metrical way.

Chapter 4

Conclusions and Future Works

4.1 Conclusions

This thesis described the processes to develop non parametrical algorithms based in the concept of permutation entropy. The starting point of the study was the reproduction of the work proposed in [1], where permutation entropy was used along with bootstrap and confidence intervals to determine similarities between EEG time series. The reproduction did not succeed to give the anticipated results and did not determine the similarities relationships as expected. However, those negative results led to more analysis of the method of permutations giving two crucial lessons. First, that the permutation rank based statistics were indeed characterizing different patterns among the different defined groups of data but when those statistics were transformed into the value of permutation entropy the characterization was lost. Thus, this taught us the second crucial lesson, that the statistics built from permutation analysis of the data would work properly for characterizing purposes if those statistics were used with another metric rather than the strict definition of permutation entropy. Since the goal was to develop a classifying algorithm, the concept of comparisons between distributions was required. Therefore, one metric fitting that requirement is the Total Variation Distance (TVD).

After learning from the failed reproduction the development of our own algorithms started. The previous chapter showed how the concepts of permutation rank based statistics were teamed with the TVD approach to develop classifying algorithms. Three methods were developed using the mentioned concepts and relying on the widely accepted L_2 norm to determine classification decisions. At first the algorithms were tested using EEG data, however the complex behavior of brain measurements gave surprising results due to the random behavior of them. Then, our own generated data was used to give better control of the expected results, and also a widely used parametric classification method (Maximum Likelihood Estimator) was introduced to play the benchmark role. As explained in more details in Chapter 3, the algorithms when tested using single distributed groups worked almost flawlessly for time series of considerable length. Also, when tested using mixed distributed groups (as a way to emulate real life time series) results were fairly acceptable and in some cases outperforming the ML approach. In summary, our work proved the permutation rank based statistics as a powerful tool to characterize underlying statistical behavior of time series, and we further proposed three simple but functional algorithms to classify time series based on that concept.

4.2 Future Works

The presented work is a good starting point to motivate additional research on related topics. Some relevant ideas came through discussion during the study that may be good research topics in the future. Among the most interesting and relevant we have:

- The presented algorithms can be improved with deeper research in the behavior of the permutation statistics and the proposition of different metric. Also, the comparison intended in our work was meant to be as simple as possible but functional, however other alternatives rather than using vector or matrices norms could be researched and proposed.

- Permutation rank based statistics proved to work for classification of time series purposes. Nonetheless, it was observed in our results that when the length of the time series is not long enough then the effectiveness of the approach decreased. Some studies could be done to extend this method to be also effective for short time series.

- From our recent results, we already found that finding a proper representative distributions within a same class is an effective starting point for classification purposes. However, the built statistics are based in Transition Probability Matrices forming a Markov chain between the possible permutations. Therefore, a way to improve what was done is to seek proper metrics to measure distances between multiple Markov probability transitions matrices, note that in each of this matrix, we have a product of multi-nomial distributions each of which is a point in a simplex.

- Through the thesis we adopted of a first order Markov chain modeling the symbol sequences using the permutation entropy framework. As seen in the analysis carried on in Appendix B, the behavior of the transitions of the permutations would be more accurately characterized if higher order Markov chain modeling is used.

- During our study we did not find time series containing equal values. Nevertheless, there could be time series containing a large amount of equal values and even consecutive. In this scenario, the permutation rank based statistics might lead to misleading results. This limitation of this approach have not been fully overcome yet.

- The same concepts used in our classification method could be used for prediction purposes, where we would be more keen to predict the trend of a time series based on the resulting ranking trends. For discrete cases a good approach to get best optimals is the Gradient Descent Search, which teamed with the permutation based ranking statistics could be a good approach for prediction problems. However, using this approach will increase the complexity of the resulting algorithm.

- We used the concept of absolute error in the decision making of our classification algorithms. However, extending the analysis to consider the conditional error might lead to better results and further increase the applicability of our work.

- In [28] a ranking-based evaluation of time series was proposed where instead of taking rankings of small windows a complete ranking was performed in the entire sequence allowing to spot local and global maximums or minimums as well as outliers. This alternative modeling approach not just captures the trend like in the permutation entropy analysis but captures the exact ordinal statistic making it a good modeling technique for estimation and prediction purposes. Also in [29] another approach using ranking statistics proved that using a global ranking approach to look for the frequencies of top rankings, reduces the cost as well as the time required to build performance models. In both cases, the ranking was not done in small windows but in the entire sequence giving different options of statistics to analyze that could be used for new research in the time series classification or prediction problems.

Bibliography

[1] Francisco Traversaro and Francisco O. Redelico. Confidence intervals and hypothesis testing for the permutation entropy with an application to epilepsy. *Communications in Nonlinear Science and Numerical Simulation*, 57:388 – 401, 2018.

[2] Bernard Marr. Every day? the mind-blowing stats everyone should read, 2018.

[3] Statista. Number of smartphone users worldwide from 2014 to 2020 (in billions), 2018.

[4] University of South Carolina. Definition and examples of time series, 2014.

[5] Jan Beran, Yuanhua Feng, Sucharita Ghosh, and Rafal Kulik. Long-memory processes, 01 2013.

[6] George Cochran. Lecture notes in mathematical statistics, August 2018.

[7] Tomasz Grecki and Maciej uczak. Multivariate time series classification with parametric derivative dynamic time warping. 42, 11 2014.

[8] Maciej uczak. Univariate and multivariate time series classification with parametric integral dynamic time warping. 33:2403–2413, 09 2017.

[9] Sally Hunsberger, Paul S Albert, Dean A Follmann, and Edward Suh. Parametric and semiparametric approaches to testing for seasonal trend in serial count data. 3:289–98, 07 2002.

[10] G.H. Chen, S Nikolov, and D Shah. A latent source model for nonparametric time series classification. 01 2013.

[11] A. Mosallam, Kamal Medjaher, and Noureddine Zerhouni. Nonparametric time series modelling for industrial prognostics and health management . 2013.

[12] Juan Vilar, Jos Vilar, and Sonia Prtega-Daz. Classifying time series data: A nonparametric approach. 26:3–28, 04 2009.

[13] Mikolaj Binkowski, Gautier Marti, and Philippe Donnat. Autoregressive convolutional neural networks for asynchronous time series. In *ICML*, 2018.

[14] Christoph Bandt and Bernd Pompe. Permutation entropy: A natural complexity measure for time series. *Phys. Rev. Lett.*, 88:174102, Apr 2002.

[15] Massimiliano Zanin, Luciano Zunino, Osvaldo A. Rosso, and David Papo. Permutation entropy and its main biomedical and econophysics applications: A review. *Entropy*, 14(8):1553–1577, 2012.

[16] M. Riedl, A. Muller, and N. Wessel. Practical considerations of permutation entropy. *The European Physical Journal Special Topics*, 222(2):249–262, Jun 2013.

[17] University of Postdam. Tocsy - toolboxes for complex systems, 2014.

[18] Mariano Matilla-Garca and Manuel Ruiz Marn. A non-parametric independence test using permutation entropy. *Journal of Econometrics*, 144(1):139 – 155, 2008.

[19] Haroldo V. Ribeiro, Luciano Zunino, Renio S. Mendes, and Ervin K. Lenzi. Complexity-entropy causality plane: a useful approach for distinguishing songs. *CoRR*, abs/1112.2316, 2011.

[20] Jianan Xia, Pengjian Shang, Jing Wang, and Wenbin Shi. Permutation and weighted-permutation entropy analysis for the complexity of nonlinear time series. 31, 07 2015.

[21] Sebastian Berger, Gerhard Schneider, Eberhard F. Kochs, and Denis Jordan. Permutation entropy: Too complex a measure for eeg time series? *Entropy*, 19(12), 2017.

[22] Luciano Zunino, Felipe Olivares, Felix Scholkmann, and Osvaldo Rosso. Permutation entropy based time series analysis: Equalities in the input signal can lead to false conclusions. 381, 04 2017.

[23] Ralph G. Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and C. E. Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: dependence on recording region and brain state. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 64 6 Pt 1:061907, 2001.

[24] David Rosenberg. Lecture notes in stein's method and applications, August 2007.

[25] Stefan Kiefer. On computing the total variation distance of hidden markov models. 04 2018.

[26] Universitat Pompeu Fabra. The bern-barcelona eeg database, 2012.

[27] Ralph G. Andrzejak, Kaspar Schindler, and Christian Rummel. Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients. *Phys. Rev. E*, 86:046206, Oct 2012.

[28] S. Rosset, C. Perlich, and B. Zadrozny. Ranking-based evaluation of regression models. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, Nov 2005.

[29] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. Using bad learners to find good configurations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 257–267, New York, NY, USA, 2017. ACM.

[30] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.

[31] Haroldo V. Ribeiro, Luciano Zunino, Renio S. Mendes, and Ervin K. Lenzi. Complexityentropy causality plane: A useful approach for distinguishing songs. *Physica A: Statistical Mechanics and its Applications*, 391(7):2421 – 2428, 2012.

[32] Ben Carroll. Learning and identification of wireless network internode dynamics using software defined radio. Master's thesis, LSU, May 2013.

[33] A.M. Fraser. *Hidden Markov Models and Dynamical Systems*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2008.

[34] Jos M. Amig, Matthew B. Kennel, and Ljupco Kocarev. The permutation entropy rate equals the metric entropy rate for ergodic information sources and ergodic dynamical systems. *Physica D: Nonlinear Phenomena*, 210(1):77 – 95, 2005.

[35] Taichi Haruna and Kohei Nakajima. Permutation complexity and coupling measures in hidden markov models. *Entropy*, 15(9):3910–3930, 2013.

[36] S. M. E. Sahraeian and B. J. Yoon. A novel low-complexity hmm similarity measure. *IEEE Signal Processing Letters*, 18(2):87–90, Feb 2011.

[37] Douglas J. Little and Deb M. Kane. Variance of permutation entropy and the influence of ordinal pattern selection. *Phys. Rev. E*, 95:052126, May 2017.

[38] Cheng-Der Fuh. The bootstrap method for markov chains /. 22, 01 1998.

[39] Jeremy Orloff and Jonathan Bloom. 18.05 introduction to probability and statistics, Spring 2014.

Appendix A

Generated Code

This appendix provides a quick summary of the main scripts developed so far is presented. All the functions have been created following the algorithms described previously and through more research. The bootstrap approach was developed using the approach described in [38]..

A.1 Algorithm to compute PE

This whole block is then put into a function called Perm_Entr(y,tau,n).

- y = rand(1,10000);
- tau = 1;
- n=3;
- state_uni=St_Uni(n); (Give the universe of possible states n!)
- state_seq = StSeq(y,n,tau,state_uni); (The main function! Will get the original series, group the data according to n and tau, identify the permutations for each window, rank the permutations from 1 to $n!$, and give a sequence of ranked permutations)
- perm_num = PermNum(state_seq, state_uni); (Knowing the sequence of states, each state is counted and stored in an array)
- perm_prob = PermProb(perm_num); (with the count of each permutation, the probability for each one is computed)
- TPM = TranProbMat(state_seq, perm_num); (Knowing the state sequences and the number of occurrence of each state a transition probability matrix is calculated)
- H = PE(perm_prob); (Having the probabilities for each permutation, the entropy equation is used)

A.2 Algorithm to compute PER

All the previous steps must be followed but instead of calling the function H, the following two functions are used:

- st=stationary(TPM);
- Hx = rate(TPM,st,state_uni);

A.3 Algorithm to compute Bootstrap with Confidence Interval

- y1 = rand(1,1000); 1st time series.
- y2 = rand(1,1000);2nd time series.
- tau = 1;
- n=3; window size
- B=1000; # of bootstrap iterations.
- (state_uni1, state_seq1, perm_prob1, H1, TPM1) = Perm_Entr(y1,tau,n); Calculate PE for time serie 1
- (state_uni2, state_seq2, perm_prob2, H2, TPM2) = Perm_Entr(y2,tau,n); Calculate PE for time serie 2
- delta = H1 - H2; Get the difference between the PE's.
- booth_H1 = Boothstrap(B, state_uni1, state_seq1, perm_prob1, TPM1); Generate bootstrap PE for time serie 1
- booth_H2 = Boothstrap(B, state_uni2, state_seq2, perm_prob2, TPM2); Generate bootstrap PE for time serie 2
- booth_delta = Booth_Delt(B,booth_H1,booth_H2); Get the difference among all the bootstrapped PE (H*1 - H*2)
- ave_booth_delta = mean(booth_delta); Calculate the mean of the booth_delta
- ult_delta = Ult_Delt(booth_delta, ave_booth_delta); Compute the pseudo standard deviation for the bootstrapped delta.

- (CI1, CI2) = CI(delta,ult_delta, ,B); Compute the upper and lower bound of the confidence interval.
- HT = Hip_Test(CI1, CI2); Check the Hypothesis Test that zero belongs to the confidence interval.

A.4 Algorithm using TVD

- tau = 1;
- n=3; window size
- B=1000; # of bootstrap iterations.
- (state_uni1, state_seq1, TPM1, stat1) = Perm_Entr_TVD(y1,tau,n); Calculate PE for time serie 1
- (state_uni2, state_seq2, TPM2, stat2) = Perm_Entr_TVD(y2,tau,n); Calculate PE for time serie 2
- TVD = TVD_Joint(state_uni1,TPM1, stat1, TPM2, stat2); Get the joint TVD.
- (booth_TPM1, booth_stat1) = Boothstrap_TVD(B, stat1, state_uni1, state_seq1, TPM1); Generate bootstrap TPM and stat. distibution for time serie 1
- (booth_TPM2, booth_stat2) = Boothstrap_TVD(B, stat2, state_uni2, state_seq2, TPM2); Generate bootstrap TPM and stat. distibution for time serie 2
- booth_TVD = Booth_Delt_TVD(B, state_uni1, booth_TPM1, booth_stat1, booth_TPM2, booth_stat2); Get the difference among all the bootstrapped TMP and stat
- ave_booth_TVD = mean(booth_TVD); Calculate the mean of the booth_TVD
- ult_delta = Ult_Delt(booth_TVD, ave_booth_TVD); Compute the pseudo standard deviation for the bootstrapped delta.
- (CI1, CI2) = CI(delta,ult_delta, ,B); Compute the upper and lower bound of the confidence interval.
- HT = Hip_Test(CI1, CI2); Check the Hypothesis Test that zero belongs to the confidence interval.

Appendix B

Generated Code

Relevant observations while running the experiments will be shown in this section. The following remarks are are complementary to the results already shown and will help to reach conclusions.

B.0.1 Transition Restriction with n = 3

When n=3 and τ=1 there are some restrictions about the states that can be reached after each permutation due to the overlapping the time shift selected gives. Therefore the transition probability matrix will have the form:

$$TPM = \begin{bmatrix} * & 0 & * & 0 & * & 0 \\ * & 0 & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 & * \\ * & 0 & * & 0 & * & 0 \\ 0 & * & 0 & * & 0 & * \\ 0 & * & 0 & * & 0 & * \end{bmatrix}$$

With the transition graphic:

This shows the nature of the transitions and that is impossible to reach some states depending in the current one. This is expected from the value of τ, there are dependencies between each state.

B.0.2 State Assignment

Next, the correspondence between each permutation with a defined state is shown in Table I.

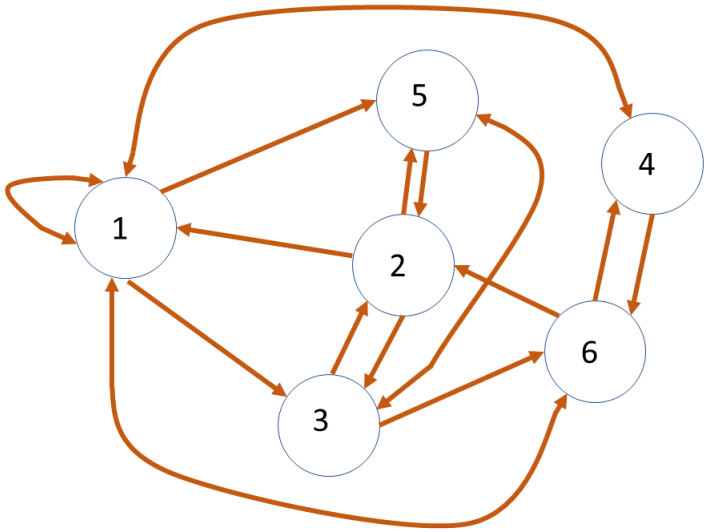

Figure B.1: State Transitions

Table B.1

State	Permutation
1	321
2	231
3	312
4	132
5	213
6	123

Table B.2

State	Pattern
1	-
2	221
3	211
4	121
5	212
6	111, 112, 222, 122

B.0.3 Equal Number Problematic

Most time series, including the EEG data used, have float type numbers which make the probability of having the exact same number repeated in one permutation window really low. However, there still a slightly chance to face that scenario and that is the reason the proposed algorithm have a mechanism to deal with identical numbers in the same permutation window. Therefore, for duplicate numbers the algorithm count as smaller the one appearing first. Since the EEG data or the Gaussian generated sequences do not present duplicated numbers new forged sequences were created. Using the Simu_Markov() function a sequence is created using only 2 by 2 and 3 by 3 matrices, forcing duplicated numbers to always appear. Using this scenarios statistical values were taken and multiple analysis performed.

B.0.3.1 2 by 2 TPM

First, two sequences were created following the transition matrices:

$$TPM1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} TPM2 = \begin{bmatrix} 0.4 & 0.6 \\ 0.8 & 0.2 \end{bmatrix}$$

with stationary distributions stat1 = [0.6667, 0.3333] and stat2 = [0.5714, 0.4286].

The simulation of the sequence following this two TPM's are sequences of 2000 points of pure 1's and 2's. This binary chains were windowed like usual to perform the permutation analysis, so from a binary chain six states have to be defined. After running the algorithm the state assignment is showed in Table II, where it can be seen that state six has four patterns associated with it and that state one never occurs.

Having constructed this scenario first we wanted to build more confidence in our algorithm. Since we have both TPM's and stationary distributions for each chain we can anticipate the probability of occurrence for each state and then compare the result with the probability calculated empirically by the algorithm. An example of how to calculate this probability is

Table B.3

State	Expected	Empirical
1	$P_1(Y=1)=0$ $P_2(Y=1)=0$	$P_1(Y=1)=0$ $P_2(Y=1)=0$
2	$P_1(Y=2)=0.079992$ $P_2(Y=2)=0.068576$	$P_1(Y=2)=0.0690$ $P_2(Y=2)=0.0655$
3	$P_1(Y=3)=0.139986$ $P_2(Y=3)=0.137152$	$P_1(Y=3)=0.1491$ $P_2(Y=3)=0.1356$
4	$P_1(Y=4)=0.120006$ $P_2(Y=4)=0.274272$	$P_1(Y=4)=0.1261$ $P_2(Y=4)=0.2766$
5	$P_1(Y=5)=0.059994$ $P_2(Y=5)=0.205728$	$P_1(Y=5)=0.0460$ $P_2(Y=5)=0.2071$
6	$P_1(Y=6)=0.600022$ $P_2(Y=6)=0.314272$	$P_1(Y=6)=0.6098$ $P_2(Y=6)=0.3152$

$$P(Y=2)=P(x_1=2)P(x_2=2|x_1=2)P(x_3=1|x_2=2)$$
$$P_1(Y=2)=0.3333*0.4*0.6=0.079992$$
$$P_2(Y=2)=0.4286*0.2*0.8=0.068576$$

Following this procedure for all the remaining states and comparing those results lead to the results presented in Table III, where we can see that the anticipated results are close to the ones calculated running the algorithm.

Now, we compute the 6 by 6 transition probability matrix of the states resulting in:

$$XS1=\begin{bmatrix}0&0&0&0&0&0\\0&0&0.7&0&0.3&0\\0&0&0&0&0&1\\0&0&0.7&0&0.3&0\\0&0&0&0.6&0&0.4\\0&0.134&0&0.14&0&0.73\end{bmatrix}\quad XS2=\begin{bmatrix}0&0&0&0&0&0\\0&0&0.4&0&0.6&0\\0&0&0&0&0&1\\0&0&0.4&0&0.6&0\\0&0&0&0.8&0&0.2\\0&0.16&0&0.27&0&0.57\end{bmatrix}$$

Calculating the stationary distributions for both matrices leading to:

$$\pi_1^\infty=[0\quad 0.0799\quad 0.14\quad 0.12\quad 0.06\quad 0.6001]$$
$$\pi_2^\infty=[0\quad 0.0617\quad 0.1248\quad 0.2503\quad 0.1862\quad 0.3762]$$

Table B.4

State	Pattern
1	321
2	331 332 221 231
3	311 322 211 312
4	232 121 131 132
5	323 212 213 313
6	233 112 213 223 333 133 222 111 122 113

Table B.5

State	Expected	Empirical
1	P(Y=1) = 0.06042	P(Y=1) = 0.0560
2	P(Y=2) = 0.135009	P(Y=2) = 0.1426
3	P(Y=3) = 0.116883	P(Y=3) = 0.1226
4	P(Y=4) = 0.152525	P(Y=4) = 0.1431
5	P(Y=5) = 0.168033	P(Y=5) = 0.1631
6	P(Y=6) = 0.367134	P(Y=6) = 0.3727

Where we can see that the values of the stationary distribution are almost identical to the anticipated state probabilities for the fist case, and similar for the second.

B.0.3.2 3 by 3 TPM

A similar experiment was held with another sequence with Transition Probability:

$$TPM3 = \begin{bmatrix} 0.3 & 0.5 & 0.2 \\ 0.4 & 0.3 & 0.3 \\ 0.1 & 0.5 & 0.4 \end{bmatrix}$$

with stationary distribution stat3 = [0.2813, 0.4167, 0.3021]

In this case we have three possible states (1, 2, and 3), and as we can see in Table IV now all states are possible and only state 1 has one pattern assigned and state 6 again is the state with more patterns assigned.

Repeating the comparison between the expected probability with the empirical results we again have close values as seen in Table V. Computing the transition matrix for this new state sequence we got:

$$XS3 = \begin{bmatrix} 0 & 0 & 0.3226 & 0 & 0.6774 & 0 \\ 0.1585 & 0 & 0.4038 & 0 & 0.4377 & 0 \\ 0 & 0.1545 & 0 & 0.610 & 0 & 0.7846 \\ 0.2654 & 0 & 0.3172 & 0 & 0.4175 & 0 \\ 0 & 0 & 0 & 0.6 & 0 & 0.5167 \\ 0 & 0.2961 & 0 & 0.2025 & 0 & 0.5 \end{bmatrix}$$

Calculating the stationary distribution for this matrix resulted in:

$$\pi_3^{\infty} = [0.0621 \quad 0.1326 \quad 0.1227 \quad 0.1547 \quad 0.1648 \quad 0.3632]$$

Once more, the values of the probability of each state is really similar to the ones derived from the stationary distribution of XS3.

Those two experiments showed that there is a correspondence using a first order Markov chain analysis even with the constant presence of repeated values in the permutations. At the beginning of the experiment we thought that the values from the marginal probabilities of the states will be different than from the ones coming from the stationary distribution of the corresponding transition matrix, but the results proved our hypothesis wrong. However, we believe that in this scenarios a first order Markov analysis is not enough to fully characterize the dynamics of the permutations due to the repetitions. Further experiments using higher order Markov analysis should be done to confirm if our hypothesis is correct.

Vita

Aldo Duarte Vera Tudela was born on June 22, 1989, in Lima, Peru. He graduated with a Bachelors in Science degree in Telecommunication Engineering in 2012 from Ponticia Universidad Catolica del Peru. He is currently pursuing a Masters of Science in Electrical Engineering at Louisiana State University, and is expected to graduate in December 2018. His research interests include wireless communications, digital signal processing, and data analysis. He has worked as a Graduate Research Assistant in the Division of Electrical and Computer Engineering under Dr. Shuangqing Wei from December 2017 to September 2018.